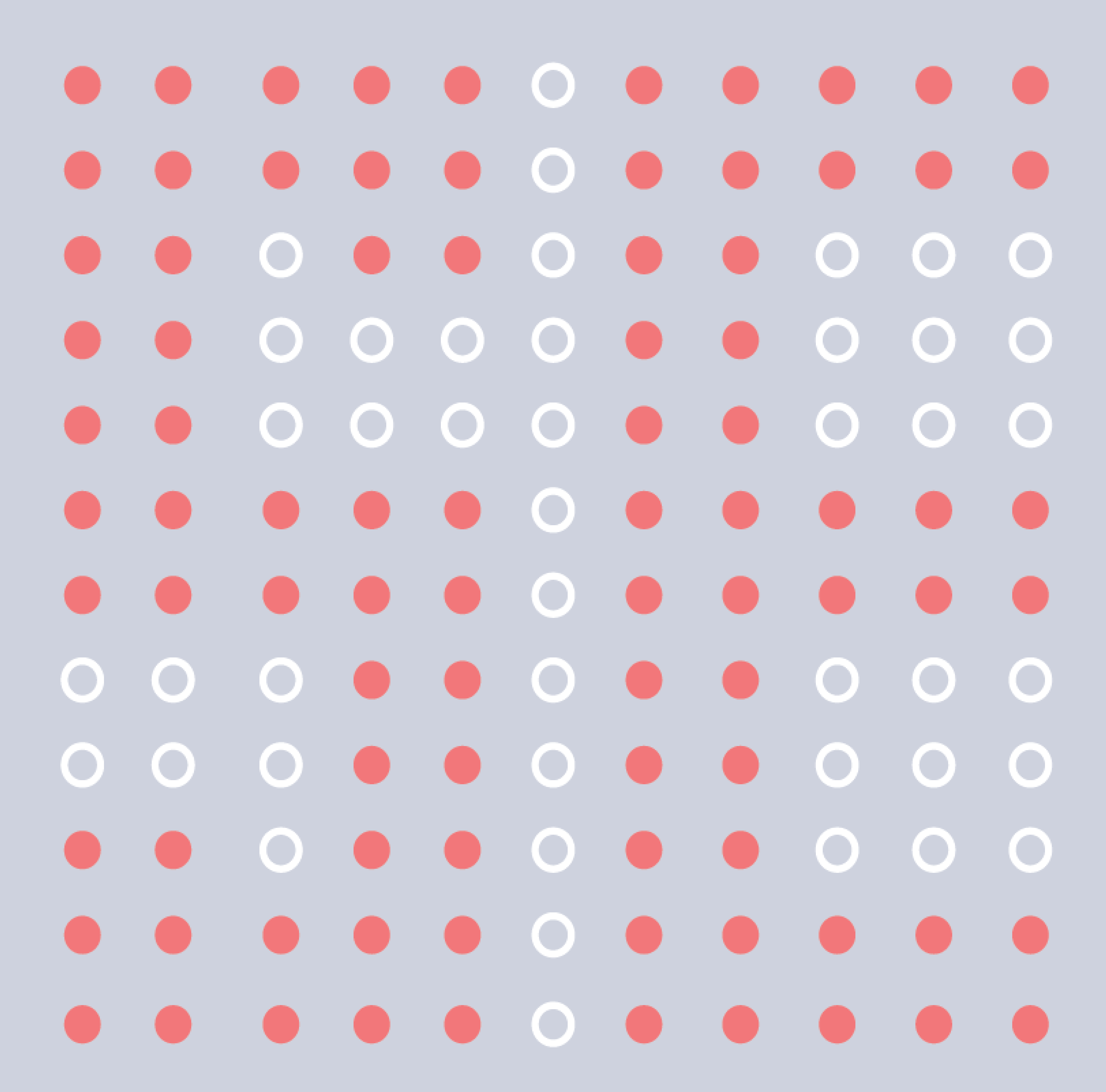


浙江省高校重点教材建设项目

姚茂群　主　编
方　敏　王玉槐　副主编

软件工程系列教材

软件测试技术与实践



软件测试技术与实践

http://www.tup.com.cn

清华大学出版社

软件工程系列教材
浙江省重点教材

软件测试技术与实践

姚茂群　主编
方　敏　王玉槐　副主编

清华大学出版社
北　京

内 容 简 介

本书以案例贯穿全书，主要介绍软件测试技术与实践的基础知识。全书共11章，不仅讲述了软件测试的基本理论和方法（单元测试、集成测试、系统和验收测试、测试用例设计和软件缺陷跟踪管理等），而且详细介绍了企业级软件测试的解决方案（负载测试、功能测试、Web站点测试和面向对象单元测试等）及测试自动化工具（QTP、LoadRunner和JIRA等）的使用。本书为读者在软件生命周期各个阶段合理选择适当的测试技术与测试工具并有效应用到项目中，提高软件的质量和可靠性提供了指导。

本书适合作为高等学校计算机相关专业软件测试课程的教材，也可作为软件测试实训、培训班的教材和软件测试人员、软件开发人员及需要了解测试知识的各级软件管理人员的参考书。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

软件测试技术与实践/姚茂群主编. --北京：清华大学出版社，2012.7
软件工程系列教材
ISBN 978-7-302-28921-0

Ⅰ. ①软…　Ⅱ. ①姚…　Ⅲ. ①软件－测试－高等学校－教材　Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2012)第107673号

责任编辑：焦　虹　顾　冰
封面设计：常雪影
责任校对：白　蕾
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市李旗庄少明印装厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**15.25　　**字　　数：**351千字
版　　次：2012年7月第1版　　**印　　次：**2012年7月第1次印刷
印　　数：1～3000
定　　价：29.00元

产品编号：044608-01

前言

foreword

随着软件危机的频频出现，其造成的损失巨大，人们越来越认识到软件质量的重要性。而软件测试就是确保软件质量的重要手段。测试是目前用来验证软件是否能够完成预期功能的唯一有效方法。软件行业对软件测试技术的要求也越来越高。

软件测试技术发展到现代，以往纯手工的系统测试已远远不能满足全面测试的需要。软件产品生命周期长，经常变更和升级，不断增加新功能，版本不断更新。除了测试修改过的模块外，每次都要重复测试有关联的模块，这样很多时候会做大量的重复工作，很难达到测试效果。另外，配置管理要求每日构建，每天都要执行“冒烟测试”。自动化测试是目前测试领域的发展方向，自动化测试能有效地降低测试成本、提高测试效率和测试覆盖面。

为了满足社会对软件测试人员的大量需求，各高等院校都相继开设了“软件测试”课程。目前，已有相应的教材出版发行，但面向应用型本科的教材则相对较少。针对应用型本科（独立学院）计算机及相关专业的教学，介绍软件的完整测试过程，使学生能根据软件测试文档实施软件测试，提高软件测试能力，培养团队协作精神，逐步积累软件测试经验，是本书编写的初衷之一。

另外，近年来中国软件（服务）外包产业发展迅速。针对软件外包产业的实训、实践，介绍软件测试项目的技术要求和测试流程，结合案例系统阐述各种常见测试技术；介绍软件自动化测试概念和方法，结合案例阐述如何使用一些常见的软件测试工具，也是本书编写的初衷之一。

本书本着应用性、实用性和够用性三项原则编写。其中，各章相对独立，可以只读其中任一章的内容，而无须其他章节做铺垫。

第1章详细介绍了软件质量的定义、软件测试的基本概念及其生命周期，使读者建立一个完整的软件测试概念。

第2章介绍了软件测试的理论方法。

第3章详细介绍了软件测试过程，包括测试计划、测试设计、测试过程模型、测试实施以及测试评估等。

第4章主要介绍了软件缺陷的定义、分类、产生原因、生命周期及其如何穿透测试等的bug的跟踪管理的相关知识。

第5章介绍了软件测试自动化和自动化测试工具等相关知识。

第6章介绍了功能测试,重点介绍了功能测试工具QTP的使用,并结合实例讲解了如何使用QTP进行Windows应用程序和Web站点的功能测试。

第7章详细介绍了面向对象单元测试的基本概念和JUnit框架及类测试。

第8章介绍了负载测试及LoadRunner。

第9章介绍了测试流程自动化,重点介绍了JIRA的使用,包括JIRA的应用、项目配置及权限等。

第10章详细介绍了软件质量保证SQA,评审与软件测试管理的相关知识。

第11章是一个完整的实际软件项目的测试案例,详细介绍了软件测试项目从测试计划、测试过程、测试用例设计、测试实现到缺陷报告和测试结果分析的全过程,涉及了功能测试和性能测试两方面。

本书为浙江省重点教材,适合作为高等学校,尤其是应用型本科计算机及相关专业“软件测试”课程的教材,也可作为软件测试实训、培训班的教材和软件测试人员、软件开发人员及需要了解测试知识的各级软件管理人员的工作参考手册。

本书由姚茂群教授主编,方敏、王玉槐副主编,参加本书编写的还有寿周翔、王李冬、安康、吕明琪、刘丹凤、王东刚等。在此,衷心感谢对本书的编写和审定提供支持和帮助的所有机构、单位和个人。由于时间仓促,加之编者水平有限,书中难免存在不妥和错误之处,恳请大家不吝赐教。

编　者

2012年4月于杭州西湖

目录 Contents

第1章 chapter 1

软件测试概述

软件测试是软件质量保证的一个手段。本章首先让读者了解软件质量的概念；然后使读者更好地理解软件测试是什么，以及软件测试的目的和意义；最后，阐述完整的软件测试概念，包括软件测试的标准和分类，以及软件测试的生命周期。

1.1 软件质量的定义

“质量(quality)”这个词，从汉语文字角度来看，是由“质”和“量”构成的，就是在质和量上的程度。量的含义比较容易理解，而质的含义相对比较复杂。“质”作为形容词，在这里，理解为事物的本质和素质。

而对于软件质量，它是一个软件企业成功的必要条件，其重要性无论怎么强调都不过分。软件质量和传统意义上的质量概念并无本质区别，只是针对软件的某些特性进行了调整。软件质量由三部分组成[1]：

(1) 软件产品的质量，即满足使用要求的程度。

(2) 软件开发过程的质量，即能否满足开发所带来的成本、时间和风险等要求。

(3) 应用领域或者业务上的质量。

高品质软件应该是相对无产品缺陷(bug free)或仅有极少量缺陷的软件。它能够准时递交给客户，并且满足客户要求。所有开销都是在预算内，是可维护的。

总的来说，软件质量具有3A特性：Accountability(可说明性)、Availability(有效性)和 Accessibility(易用性)[1]。

(1) 可说明性：用户可以基于产品或者服务的描述和定义(如市场需求说明书、功能说明书等)加以使用。

(2) 有效性：产品或服务对于客户的需求是否保持有效，如99.99%的功能或服务有效就可以说达到质量要求。

(3) 易用性：对于用户来说产品或服务应非常容易使用，且功能非常有用(如确认测试用户可用性测试等)。

对于广义的软件质量，又是由产品质量、过程质量和商业环境质量这三者决定的。下面分别对这三者作进一步的介绍[1]。

1. 产品质量

产品质量是人们实践产物的属性和行为，是可以辨识的，并能进行科学的描述。可以通过一些方法和人类活动，来改进产品的质量。软件产品质量一般体现在以下几个方面。

- 功能性(functionality)：软件所实现的功能达到它的设计规范和满足用户需求的程度。
- 可用性(usability)：对于一个软件来说，用户学习、操作、准备输入和对输出结果的理解所做努力的程度，如安装简单方便，容易使用；界面友好，并能适用于不同特点的用户，包括对残疾人、有缺陷的人能提供产品使用的有效途径或手段。
- 可靠性(reliability)：是用户使用的根本。指在规定的时间和条件下，软件所能维持其正常的功能操作、性能水平的程度。
- 性能(performance)：在指定的条件下，用软件实现某种功能所需的计算机资源(包括内存大小、CPU 占用时间等)的有效程度。
- 容量(capacity)：系统的接受力、容纳或吸收的能力，或某功能的最大量或最大限度，有时需要确定系统的需求所能容纳的最大量，所能表现的最大值(如 Web 系统所能承受多少并发用户访问，会议系统可以承受的与会人数等)。
- 可测量性(scalability)：系统某些特性可以通过一些量化的数据指标描述其当前状态或理想状态。
- 可维护性(service manageability)：在一个运行软件中，当环境改变或软件发生错误时，进行相应修改所做努力的程度。
- 兼容性(compatibility)：软件从一个计算机系统或环境移植到另一个系统或环境的容易程度，或者是一个系统和外部条件共同工作的容易程度。
- 可扩展性(extensibility)：指将来功能增加，系统扩充的难易程度或能力。

2. 过程质量

探索复杂系统开发过程的秩序，按照一定规程工作，可以较合理地达到目标。规程由一系列活动组成，形成方法系统，建立严格的工程控制方法，要求每一个人都要遵守工程规范。

3. 软件在商业环境中所体现的质量

开发软件的目的主要是要投放市场，其质量的表现最终还要在其生存的商业环境中体现出来。软件在商业环境中的表现好坏，不一定与产品质量和软件开发过程质量保持同步。一款好的软件产品不一定获得好的市场，因为软件产品会涉及与其商业应用环境相关的一些因素，包括客户培训，向市场发布的日程安排、商业风险评估、产品的客户、维护和服务成本等。

软件产品投放到市场时，要考虑培训的周期和用户的习惯意识。软件发布的时间会受到市场的影响，或者说，制定一个合适的软件发布时间，对打开市场有很大的影响。

1.2 软件测试的基本概念

1.2.1 软件测试的定义

“软件测试”在 ISO 9000 中是这样被定义的：软件测试是一种基于机器的，对代码执行测试，确认测试的活动。大部分的软件公司和组织对它的定义都相对狭义，这样测试就被视为针对软件编码执行相对测试用例的活动了。扩展后的测试定义是：测试是发现并指出软件（包含软件经建模、需求、设计等阶段所产生的大量输出工件）中存在缺陷的过程，这个过程指明和标注问题存在的正确位置，详细记录导致问题出现的操作步骤，及时储存当时的错误状态，通过以上内容的组合，测试后问题能够准确再现，测试定义如图 1-1 所示。

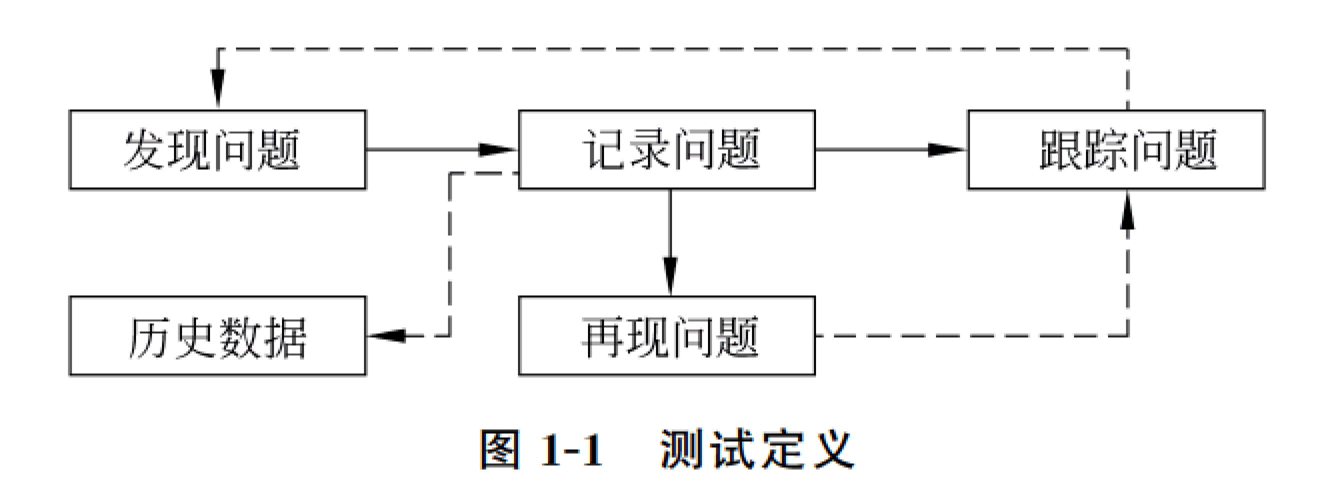


图 1-1 测试定义

图中，各阶段的含义如下：

（1）发现问题为第一个行为，表示发现软件中存在的问题。

（2）记录问题为第二个行为，表示通过“发现问题”行为操作指明和标注问题存在的正确位置，详细记录导致问题出现的操作步骤，及时储存当时的错误状态。

（3）跟踪问题为第三个行为，表示通过记录问题行为操作来跟踪和控制出现过的问题，直到问题关闭。由“再现问题”引出的虚线代表着两个行为之间的关联是隐性存在的。

（4）再现问题来源于记录问题，表示该过程只是为了演示曾经出现过的错误现象。值得注意的是，有些错误现象可能没有办法再现，例如，因某次操作引起了操作系统的内存出错异常，但是在下一次演示的过程中不一定会再出现。虽然有这样的事情发生，但是可以肯定错误还是存在的，只不过没有触发相应的条件。

（5）历史数据从记录问题中挑选相同属性的错误，提高下次测试的针对性。因为是虚框，所以可以视为过程的一种补充。

相当一部分测试理论认为跟踪问题不包括在软件测试中，属于软件开发者的事情。编者认为让开发者去跟踪问题是很荒谬的，这样大部分的错误都会在项目的逐渐扩大中丢失。只有加入了跟踪问题，软件测试才是完美的，换一句话说，就是让专业测试人员去跟踪错误。注意，跟踪问题在整个测试过程中是比较关键的。

那么，软件测试是否包含了修复？很显然软件测试和修复都是不同意义的行为过程，最能体现修复行为的是调试和修正。但是在实际的某些测试行为中这个概念比较容易混淆，例如，在以程序员为主角的单元测试中，程序员的工作就复合了测试、修复两种

行为,似乎修复就包含于测试中了。实际上,测试和修复是两种相互独立的行为过程,只是在同一个人身上编码、测试角色进行了转换。软件测试和软件修复只可能是某种意义上的重合,但却是两种截然不同的行为。

更多专家认为软件测试除了要考虑测试结果的正确性之外,还应关心程序的效率、可适用性、维护性、可扩充性、安全性、可靠性、系统性能、系统容量、可伸缩性、服务可管理性、兼容性等因素。随着人们对软件测试更广泛,更深刻的认识,可以说对软件质量的判断决不限于程序本身,而是整个软件研制过程。

不管怎么定义软件测试,基本的结论是一致的,即软件测试是为了发现软件产品所存在的任何意义上的软件缺陷(bug),从而纠正这些软件缺陷,使软件系统更好地满足用户的需求。那么,什么是软件缺陷呢?

1.2.2 软件缺陷的定义和种类

对于软件存在的各种问题,人们都用“软件缺陷”这个词,在英文中用一个不大贴切,但已专用的词 bug(臭虫)来表示。实际和“缺陷(bug)”一词相近的词还有很多,如缺点(defect)、偏差(variance)、谬误(fault)、失败(failure)、问题(problem)、矛盾(inconsistency)、错误(error)、毛病(incident)、异常(anomy)。但习惯上还是使用 bug 这个词,它包含了一些偏差、谬误或错误,更多地表现在功能上的失败(failure)和实际需求的不一致,即矛盾(inconsistency)。

软件缺陷指计算机系统或者程序中存在的任何一种破坏正常运行能力的问题、错误或者隐藏的功能缺陷、瑕疵。缺陷会导致软件产品在某种程度上不能满足用户的需要。IEEE Standard 729-1983 对软件缺陷下了一个标准的定义[1]。

(1) 从产品内部看,软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题。

(2) 从外部看,整体缺陷是系统所需要实现的某种功能的失效或者违背。

软件缺陷就是软件产品中存在的问题,最终表现为用户所需要的功能没有完全实现,没有满足用户的需求。

软件缺陷表现的形式有多种,不仅仅体现在功能的失效、放慢,还体现在其他方面。软件缺陷的主要类型有:

① 功能、特性没有实现或部分没有实现。

② 设计不合理,存在缺陷。

③ 实际结果和预期结果不一致。

④ 运行出错,包括运行中断、系统崩溃、界面混乱。

⑤ 数据结果不正确、精度不够。

⑥ 用户不能接受的其他问题,如存取时间过长、界面不美观。

美国商务部国家标准和技术研究所(NIST)进行的一项研究表明,软件中的缺陷每年给美国经济造成的损失高达几百至上千亿美元。说明软件中存在的缺陷所造成的损失是巨大的,从反面又一次证明软件测试的重要性。如何尽早彻底地发现软件中存在的

缺陷是一项非常复杂、需要创造性的工作。同时，软件的缺陷是软件开发过程中的重要属性，反映软件开发过程中需求分析、功能设计、用户界面设计、编程环节所隐含的问题，也为项目管理、过程改进提供了许多信息。

以上是缺陷的基本概念和主要类型，这里仅作一个简单的介绍，详细讨论见第4章。

1.2.3 软件测试的分类

软件测试的技术和方法多种多样。本节分别从测试范围、测试目的、测试对象和测试内容等方面进行分类。

1. 按测试范围分类

- 单元测试(unit testing)；
- 组件测试(component testing)；
- 集成测试(integration testing)；
- 系统测试(system testing)；
- 验收测试(acceptance testing)；
- 安装测试(installation testing)。

2. 按测试目的分类

- 正确性测试(correctness testing)，又可分为白盒测试(white-box)和黑盒测试(black-box)；
- 性能测试(performance testing)；
- 可靠性测试(reliability testing)，又可分为强壮性测试(robustness，strong testing)；异常处理测试(exception handling testing)和负载测试(stress，load testing)。
- 安全性测试(security testing)。

3. 按测试对象分类

- 单元测试(unit testing)；
- 组件测试(component testing)；
- 模块测试(module testing)；
- 程序测试(program testing)；
- 系统测试(system testing)；
- 文档测试(document testing)。

4. 按测试内容分类

(1) 回归测试：是指修改了旧代码后，重新进行测试以确认修改没有引入新的错误或导致其他代码产生错误。自动回归测试将大幅降低系统测试、维护升级等阶段的成本。

(2) 功能测试：是对产品的各功能进行验证，根据功能测试用例，逐项测试，检查产

品是否达到用户要求的功能。

（3）负载测试：是一种性能测试指数据在超负荷环境中运行，程序是否能够承担。

（4）压力测试：通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。通俗地讲，压力测试是为了发现在什么条件下用户的应用程序的性能会变得不可接受。

（5）性能测试：测试软件测试的性能，包括负载测试、强度测试、数据库容量测试、基准测试以及基准测试。

（6）强度测试：强度测试是一种性能测试，用于测试在系统资源特别低的情况下软件系统的运行情况。这类测试往往可以书写系统要求的软硬件水平要求。

（7）易用性测试：是指用户使用软件时是否感觉方便，例如，是否最多点击鼠标三次就可以达到用户的目的。

1.2.4 软件测试的原则

软件测试的基本原则是站在用户的角度，对产品进行全面测试，以求尽早、尽可能多地发现缺陷，并负责跟踪和分析产品中的问题，对不足之处提出质疑和改进意见。零缺陷是一种理想，足够好是测试的原则。

在软件测试过程中，应注意和遵循的原则可以概括为以下 10 项。

（1）所有测试的标准都是建立在用户需求之上。软件测试的目标在于揭示错误。测试人员要始终站在用户的角度去看问题，系统中严重的错误足以导致程序无法满足用户需求。

（2）软件测试必须基于“质量第一”的思想去开展各项工作。当时间和质量冲突时，时间要服从质量。

（3）事先定义好产品的质量标准。只有建立了质量标准，才能根据测试的结果，对产品的质量进行分析和评估。同样，测试用例应确定预期输出结果。如果无法确定测试结果则无法进行校验。必须用事先精确对应的输入数据和输出结果来对照检查当前的输出结果是否正确，做到有的放矢。

（4）软件项目一启动，软件测试也就开始，而不是等程序写完才开始进行测试。在代码完成之前，测试人员要参与需求分析、系统或程序设计的审查工作，而且要准备测试计划、测试用例、测试脚本和测试环境。测试计划要求在需求模型一完成就开始，详细的测试用例定义可以在设计模型被确定后开始。

（5）穷举测试是不可能的。即使一个大小适度的程序，其路径排列的数量也非常大，因此在测试中不可能运行路径的每一种组合。然而，充分覆盖程序逻辑，并确保程序设计中使用的所有条件是有可能的。

（6）第三方进行测试会更客观、更有效。程序员应避免测试自己的程序，为达到最佳的效果，应由第三方来进行测试。心理状态是测试自己程序的障碍，对需求规格说明的理解产生的错误也很难在程序员本人测试时被发现。

（7）软件测试计划是做好软件测试工作的前提。在进行实际测试之前，应制定良好的、切实可行的测试计划并严格执行，特别要确定测试策略和测试目标。

(8) 测试用例是设计出来的,不是写出来的。要根据测试的目的,采用相应的方法去设计测试用例,从而提高测试的效率,更多地发现错误,提高程序的可靠性。除了检查程序是否做了它应该做的事,还要看程序是否做了它不该做的事。不仅应选用合理的输入数据,对于非法的输入也要设计测试用例进行测试。

(9) 对主观错误较多的程序段,应进行更深入的测试。一般来说,一段程序中已发现的错误数越多,其中存在的错误概率也就越大。

(10) 重视文档,妥善保存一切测试过程文档。测试计划、测试用例、测试报告都是检查整个开发过程的主要依据,有利于今后流程改进,同时也是测试人员的智慧结晶和经验积累。对新人或今后的工作都有指导意义。

除了这10项原则之外,在测试当中,还有许多注意事项或经验:

- 应当把“尽早和不断地测试”作为测试人员的座右铭。
- 回归测试的关联性一定要引起充分的注意,修改一个错误而引起更多错误出现的现象并不少见。
- 测试应从“小规模”开始,逐步转向“大规模”。最初的测试通常把焦点放在单个程序模块上,进一步测试的焦点则转向在集成的模块簇中寻找错误,最后在整个系统中寻找错误。
- 不可将测试用例置之度外,排除随意性。特别是对做了修改之后的程序进行重新测试时,如不严格执行测试用例,将有可能忽略由修改错误而引起的新错误。
- 必须彻底检查每一个测试结果。事实上有相当一部分最终发现的错误是在早期测试结果中遗漏的。
- 一定要注意测试中的错误集中发生现象,这和程序员的编程水平和习惯有很大的关系。
- 对测试错误结果一定要有一个确认的过程。一般由A测试出来的错误,一定要有B来确认,严重的错误可以召开评审会进行讨论和分析。

1.3 软件测试的目的

在了解了软件测试的定义以后,很容易就会联想到软件测试的目的就是为了找出被测试软件中所有存在的错误。这样的说法并不错误只是比较理想化,事实上,不可能发现所有的问题。Grenford J. Myers罗列了四条测试观点:

(1) 软件测试是为了发现错误而执行程序的过程。

(2) 测试是为了证明程序有错,而不是证明程序无错误。

(3) 一个好的测试用例是在于它能发现至今未发现的错误。

(4) 一个成功的测试是发现了至今未发现的错误的测试。

软件测试不以发现错误为唯一目的,查不出错误的测试并非没有价值。通过分析错误产生的原因和错误的分布特征,就可以帮助发现当前所采用的软件过程的缺陷同时加以改进。同时,这种分析也能帮助设计出有针对性的检测方法,改善测试的有效性。没有发现错误的测试也是有价值的,整个测试过程本身就是评定测试质量的一种方法。如

果测试过程持续增长，在运行多次后而未发现软件错误，这样多少都可以得出这样的结论：被测试软件已经完美了，或者就是需要放弃这一无法正常工作的测试过程而重新构建了。因为存在针对性，所以软件测试存在多种目的，其中最重要的三条为：

（1）证明所做的是客户所需的。

（2）确保编码人员正确理解设计的意图。

（3）通过回归测试来保证目前运行的程序在将来仍然可以正常工作。

大部分的程序编码人员都是极其认真负责的，事后统计证明因为程序编码人员编码而导致的错误只是整个软件出错比例中很小的一块，大部分的软件错误是因为对需求不了解就设计或者是编码人员误解了设计意图而引发的。需要加强与客户代表的交流，欢迎客户代表参与到开发过程中，时刻纠正在开发过程中因为相互不了解而导致的问题。这里客户代表就是测试人员，他用自己的业务知识和对业务的了解来判断我们提供的设计是否正确。针对编码人员的误解，使用整套被大家所承认的、有能力准确表达设计意图的符号语言（如 UML 语言）来避免上面出现的情况。编码人员根据这种符号语言正确的编写代码，测试人员则根据这种符号语言测试编码人员编写的代码是否符合设计要求。

开发过程中出现错误是无法避免的，需要认真地对待它。搜集所有发现的错误，剔除虚假的和无效的，并对它们进行归类，在确认错误已经被关闭的前提下重新对其进行一次有针对性的测试。也可以这么说，只对软件进行一次测试作用是有限的，过了一段时间这次测试就会完全失效。因为软件开发是迭代的，每一次都会有新的对象加入，所以需要进行多次测试来确保测试后的程序仍然可以正常运行。

在多年的实际测试工作中，我们发现：随着软件系统规模和复杂度的不断增加，对软件系统进行较全面的测试，甚至是部分全面的测试都变得不可能；同时，软件系统的开发具有时限性，开发时间过长则容易失去或减小系统的作用；从经济性考虑，很多用户只提供有限的资源（时间、人力和物力）进行软件系统开发，而接受系统存在部分影响不大的问题，以获取较大的性价比。

因此，在商业软件测试活动中，软件测试的目的是：利用有限的资源，尽可能地发现对用户造成重大影响的问题。

1.4 软件测试的生命周期

本节从软件测试的纵向和横向两方面来分析软件测试的生命周期，并说明各种测试活动在整个软件开发过程中的位置[2]。

1.4.1 软件测试的纵向过程

从过程的观点来考虑整个测试过程的话，在软件工程环境中的测试是顺序实现的单元测试、集成测试、确认测试、系统测试四个纵向步骤的序列，这些步骤可用图 1-2 表示。

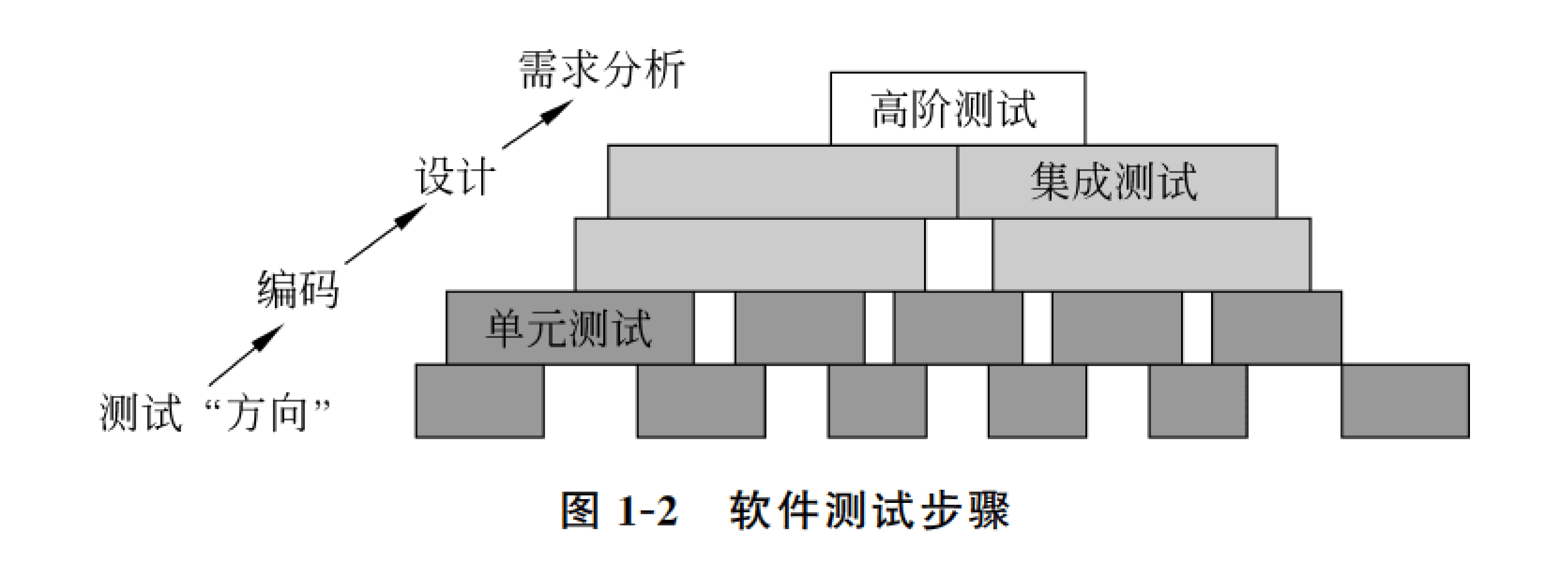


图 1-2 软件测试步骤

1. 单元测试

单元测试的特点是开始测试着重于每一个单独的模块,以确保每个模块都能正确执行。单元测试一般与单元开发同步执行,普遍的做法是在编写完一组功能代码(函数或者面向对象的类)后,马上编写单元测试代码对功能函数进行测试。当然,也有先编写测试代码再进行单元代码开发的实例。单元测试大量地使用白盒测试技术。白盒测试检查每一个控制结构的分支以确保完全覆盖和最大可能的错误检查,所以单元测试应该由程序员来完成,可以把单元测试看作是编码工作的一部分。单元测试可以借助专门的单元测试工具,如针对 Java 的 JUnit、针对 C++ 的 CppUnit 工具、DotNet 环境下的 NUnit 工具。

2. 集成测试

各个单元模块装配或集成在一起形成完整的软件包,集成测试解决的是与验证和程序构造相关的问题,应首先集成风险程度最高的模块或者位于关键路径的模块,以便能够尽早测试出这些模块是否存在问题。在集成过程中使用最多的是黑盒测试技术,为了保证一些大分支的覆盖,也会用一定数量的白盒测试技术。集成测试一般由程序员完成。

3. 确认测试

在软件集成完毕后,必须进行确认测试(高阶测试),确认软件是否符合用户的需求。确认测试提供了对软件符合所有功能、行为、性能需求的最后保证。在确认测试中,只使用黑盒测试技术,可以由最终用户参与完成。很多情况下,确认测试不被作为一个单独的测试阶段被描述,可以作为集成测试的最后阶段。

4. 系统测试

经过确认测试后,必须和其他系统元素(硬件、人员、数据库)结合在一起,由系统测试来验证所有的元素能否正确地结合在一起,满足整个系统的功能和性能要求。

事实上,软件测试的目的是发现错误。为了达到这个目的,需要计划和执行一系列的测试步骤——单元测试、集成测试、确认测试和系统测试。实践证明,测试越早介入,就越能有效降低软件开发的成本。更高质量的软件需要更系统化的测试方法,测试活动贯穿整个软件开发过程。

1.4.2 软件测试的横向过程

每一个测试阶段，从横向来看也有一个完整的过程，跟软件开发过程的线性顺序模型相似，如图 1-3 所示。一般包含的步骤如图 1-3 所示。

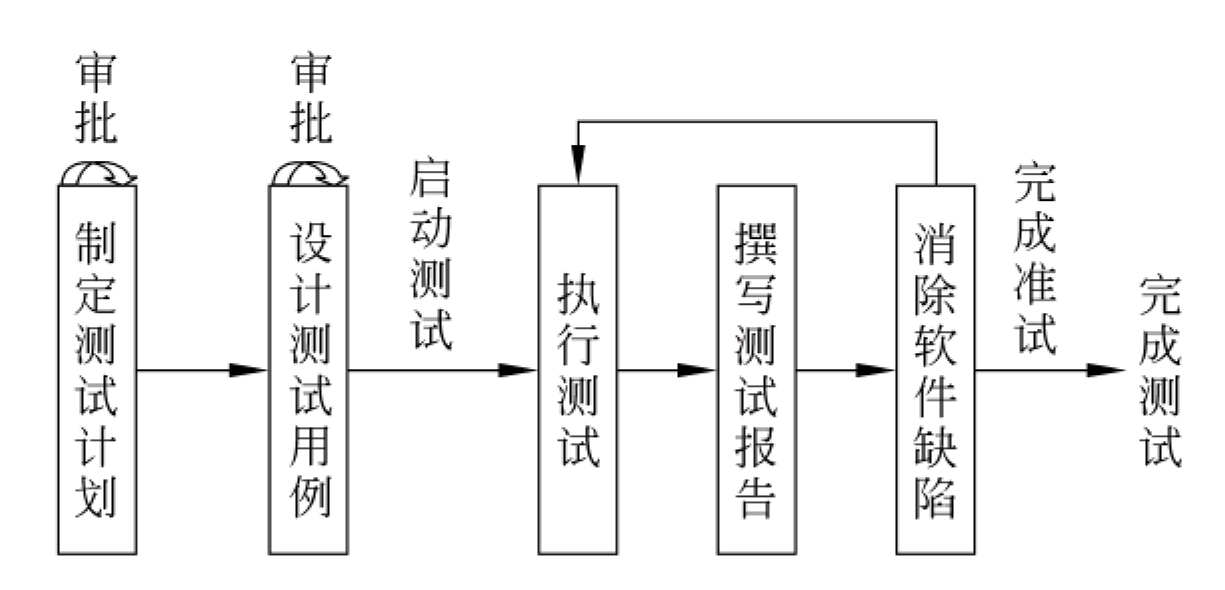


图 1-3 软件测试横向过程

1. 测试计划

测试从需求分析开始介入，测试人员参与需求的分析活动，确定测试的需求。测试计划在需求分析完成后，程序修改完毕前准备。制定测试计划主要考虑测试需求及测试进度，即需要验证什么功能需求点，采用什么测试策略，描述目前在进行哪一阶段的测试(单元测试、集成测试和系统测试)以及每个阶段进行的测试种类(功能测试、性能测试、压力测试等)。

- 输入：项目计划和测试需求。
- 输出：测试计划。

2. 测试分析与设计

测试设计在程序修改完毕前或程序修改完毕后进行。根据测试内容，分别准备合适的测试用例。测试用例基本要素有：目的、前提条件、输入数据或动作、期望的结果等。

- 输入：测试计划。
- 输出：新增测试用例及测试数据，原有回归测试数据集，真实数据。

3. 测试开发

测试开发一般与测试设计并行。测试开发主要工作为搭建测试环境，准备运行测试任务的脚本，准备检查测试结果的脚本。

- 输入：测试计划、测试用例。
- 输出：测试环境、测试脚本、验证脚本。

4. 测试执行与评估

测试执行在程序修改完毕后，可执行代码提交后执行。具体步骤如下：

① 建立测试系统。

② 准备测试过程。

③ 运行初始化过程。

④ 执行测试。

⑤ 从终止的测试恢复。

⑥ 验证预期结果，测试不通过，反馈回给编码人员修改。代码修改重新提交后，返回步骤②。

⑦ 调查突发结果。

⑧ 记录缺陷日记。

⑨ 回顾测试日记。

⑩ 评估测试需求的覆盖率。

⑪ 分析缺陷。

⑫ 决定是否达到完成测试的标准。

- 输入：测试用例、测试环境、测试脚本、验证脚本。
- 输出：测试实况记录、测试报告、缺陷报告。

1.5 软件测试与软件开发生命周期

本节概述一些常用的软件开发生命周期模型，并且特别强调了软件测试在每一个周期中所起到的作用[3]。不管使用何种生命周期模型对软件进行开发，软件都要接受测试。质量、功能都很完美的软件产品需要在其软件开发的生命周期中尽可能早地进行测试，并且无论发生什么变故，都要进行完善的回归测试。

1.5.1 顺序生命周期模型

软件开发生命周期模型以需求定义为开端，以对需求的正式验收作为终结。传统意义上，被用于软件开发生命周期的模型应该是顺序的，并且开发过程的各个阶段都经过完善的定义。通常用V型生命周期模型（见图1-4）和瀑布生命周期模型（见图1-5）来表示这种顺序的开发过程。而事实上，这两种生命周期模型有许多不同的形态，将不同的阶段引入生命周期模型，并在不同阶段之间设立边界。图1-4和图1-5所示的生命周期模型的各个阶段是经过许多富有经验的开发者经过反复实践得来的。

1. 需求分析阶段

需求分析阶段主要是收集并分析用户的需求，并且根据软件需求建立完整而明确的需求说明书。

2. 概要设计阶段

概要设计阶段主要是针对用户需求的软件结构将会被设计，并确定软件内部各个部件的相关联系。

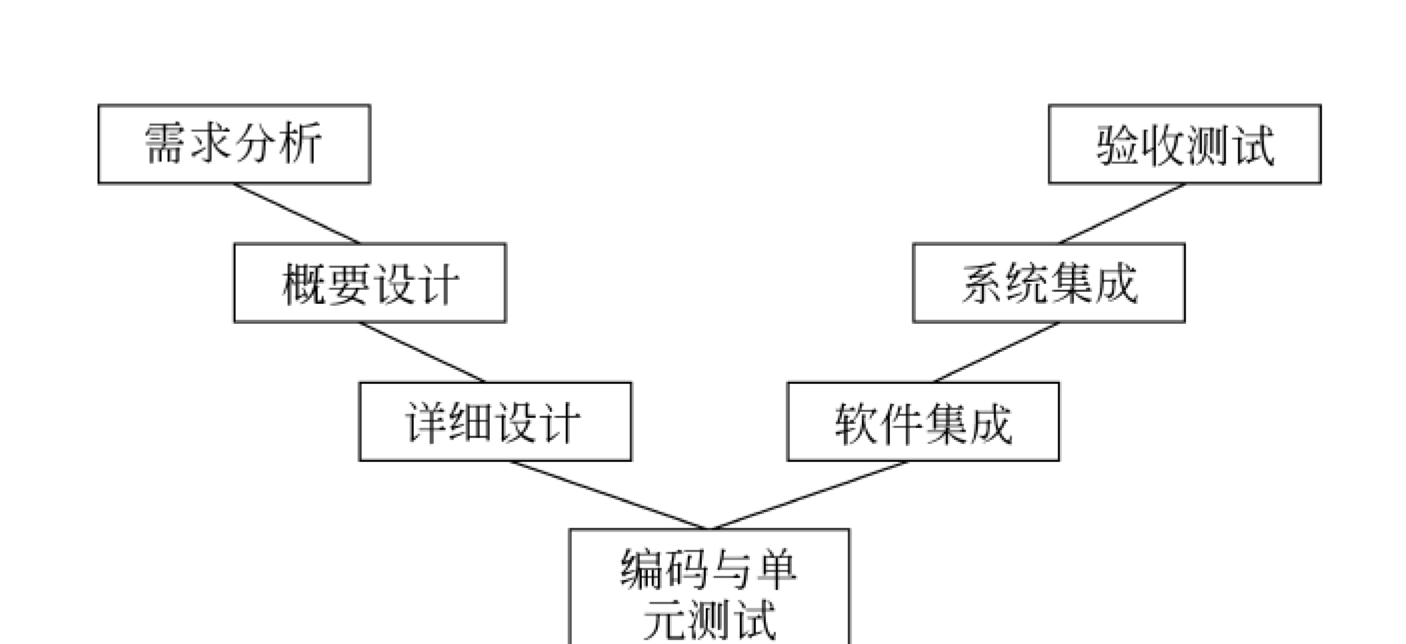


图 1-4　V 型生命周期模型

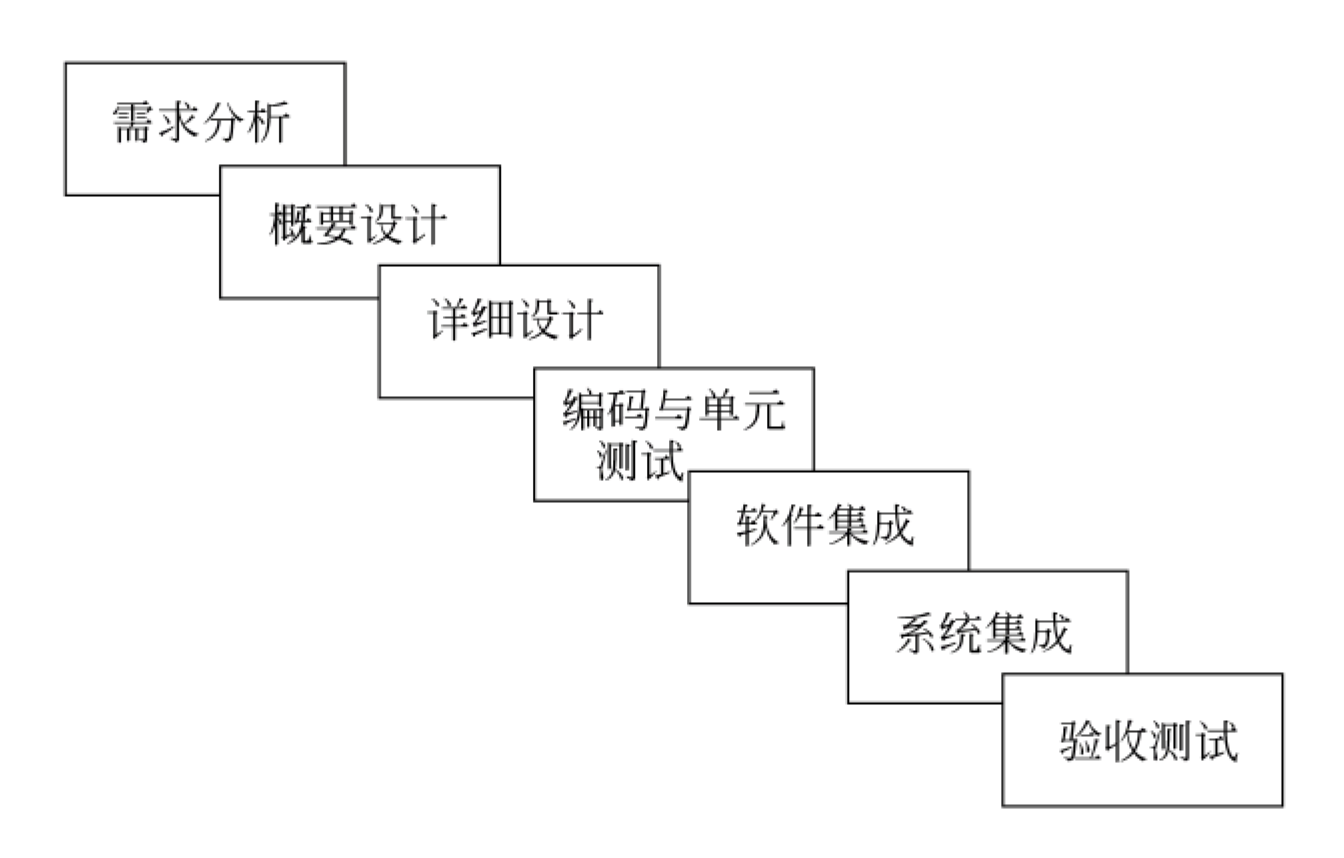


图 1-5　瀑布生命周期模型

3. 详细设计阶段

在详细设计阶段，软件各个部件的执行功能将被详细说明。

4. 编码与单元测试阶段

编码与单元测试阶段将对软件的各个部件进行编码，并且进行单元测试以确定各个部分确实执行了详细设计阶段所制定的目标。

5. 软件集成阶段

软件集成阶段被测试过的各个部件被逐渐集成起来测试直到构成一个完整的软件。

6. 系统集成阶段

系统集成阶段将软件程序集成起来，构成产品并进行测试。

7. 验收测试阶段

验收测试阶段将进行测试以验证软件确实完整地执行了用户的需求。

1.5.2 渐进开发生命周期模型

顺序模型只是代表着一种理想化的软件开发模型。实际上出于种种原因,例如需求的多变性和为应付过长的开发时间而开发临时系统的需要,还有其他生命周期模型会被采用。

现在,以渐进开发的迭代生命周期(见图 1-6)为例,来看一下其他生命周期模型。软件开发所具有的一个问题就是用户急需软件产品,但是开发者却要花费很长的时间去完整地进行开发。那么取一个折中的解决方法就是节省一些时间,但在功能上打一点折扣——开发出一个功能有所缩减的试用版给用户,作为完整版发布前的一个跳板;也可以将这个跳板作为软件减少软件开发风险的一种方式。

在软件开发中,通常将这种方法称为渐进开发或是执行阶段。与之相对应的生命周期模型被称为渐进开发生命周期模型。在渐进开发的生命周期之中,每一个独立的阶段都将遵从 V 型和瀑布型生命周期模型。

每一个软件的发布都会经过验收测试,以证明软件的各个部分所构成的整体确实实现了需求。但是每个阶段的测试和集成都会耗费大量的时间和精力。由于过多的开发周期会增加成本,耗费时间,所以应该经过认真估算,尽早地规划好到底应该使用多少个周期来进行软件的开发。

在早期开发出来的产品没有任何的实用价值,只是作为下一步开发的一个原型。这些原型仅仅是用来满足、核对用户关键需求所走的一个捷径。可是如果其中缩减了文档的书写和对软件的测试,那么就有必要将这个原型抛弃并从下一个阶段开始重新设计。因为一个缺乏质量的原型不可能给下一步的开发打下一个好基础。

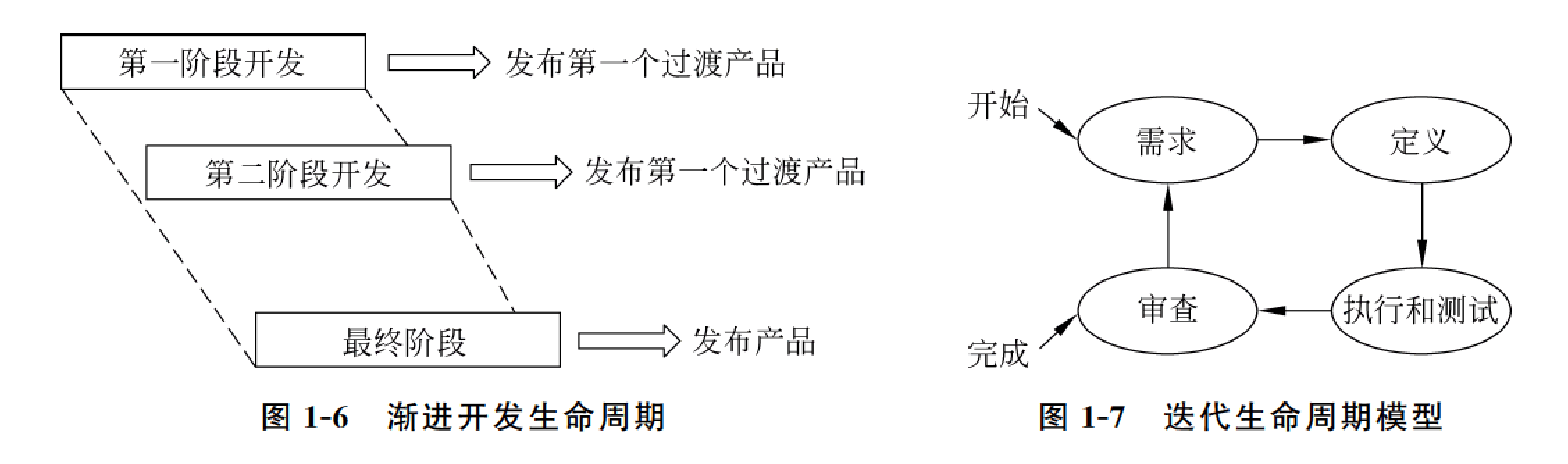


图 1-6 渐进开发生命周期 **图 1-7 迭代生命周期模型**

1.5.3 迭代生命周期模型

迭代生命周期模型并不是一开始就完全适应需求,而是根据说明先开发软件的部分可执行部件。这些部件要求能够通过审查以确定更进一步的需求。一个迭代周期模型由四个连续部分组成,如图 1-7 所示。

1. 需求分析阶段

在需求分析阶段,主要是收集并分析用户的需求,并且制定这个迭代模型最终并且

完整的需求说明书。

2. 定义阶段

在定义阶段，设计出适应需求的软件解决方案。这有可能是一个全新的设计，也有可能是原来设计的一个延伸。

3. 执行、测试阶段

在执行、测试阶段，将对软件进行编码、集成并进行测试。

4. 审查阶段

在审查阶段，将对软件进行评估，对目前的需求进行审查，并对其进行修改和更新。

在迭代模型的每一个周期，都要作出一个决定：要将编写出的软件抛弃，还是作为下一个周期的起点。如果软件完全满足了需求，那么就可以进行发布，否则就是一个失败的开始。

迭代生命周期模型可以说是采用了一种连续逼近的方式来制作软件。将这种方式类比成一个连续逼近最终解决方案的数学模型，那么这种方式能否成功的关键就是多快能够形成解决方案。

也许经过不断的类比也不能找到解决方案，而迭代的过程不是在可行的解决方案周围盘旋就是逐渐远离目标。而且需要迭代的次数过于庞大会使软件开发变得不切实际，在很多软件开发的过程中都能发现这个问题。

成功运用迭代软件生命周期模型来开发的关键就是要严格按照需求，并且根据需求对每个周期制造出来的各版本软件进行验收(包括测试)。迭代模型的前三个阶段就好比是简化版本的 V 型模型或瀑布模型。迭代模型中的每一个周期所编写出来的软件都要为软件的测试、系统测试和验收进行单元测试。在迭代模型中软件的开发经历了多少个这样的周期，就要进行多少次这样的测试。

第 2 章



软件测试方法

软件的多样性为软件测试带来了巨大挑战。新软件的发布、新技术的涌现,造成了许多新的测试问题。一个程序员面对各种可能的平台、编程语言和用户,可能会编写出的成百上千行代码,给软件测试带来了巨大的挑战。所以,软件测试是一个十分艰巨的任务。

为了检验开发的软件是否符合规格说明的要求,测试活动可以采用各种不同的测试手段和策略,如静态测试(static testing)与动态测试(dynamic testing)、黑盒测试(black-box testing)与白盒测试(white-box testing)、自顶向下的集成测试与自底向上的测试以及不同的测试步骤(如单元测试、集成测试、系统测试等)[4]。

21 软件测试方法概述

软件测试的方法多种多样,可从不同角度进行分类:从是否需要执行被测软件的角度,可分为静态测试和动态测试;从软件测试用例设计方法的角度,可分为黑盒测试和白盒测试;从软件测试的策略和过程的角度,可分为单元测试、集成测试、确认测试、系统测试和验收测试等。

1. 从是否需要执行被测软件的角度分类

从是否需要执行被测软件的角度,软件测试可分为静态测试和动态测试。

(1) 静态测试就是通过对被测程序的静态审查,发现代码中潜在的错误,一般采用人工方式脱机完成,也可借助静态分析器自动进行检查,但不要求运行程序本身。

(2) 动态测试是通常意义上的测试,即必须运行被测软件。动态测试的对象是能够由计算机真正运行的程序,动态测试包括黑盒测试和白盒测试。

2. 从软件测试用例设计方法的角度分类

从软件测试用例设计方法的角度,软件测试可分为黑盒测试和白盒测试。

黑盒测试是一种从用户角度出发的测试,又称为功能测试。它把被测程序当做一个黑盒,忽略程序内部的结构特性,依靠程序功能需求规格说明书,在只知道程序功能的情况下(即程序输入和输出之间关系)确定测试用例。简单来说,若测试用例的设计是基于

软件功能，目的是检查软件各功能是否实现，并检查其中的功能错误，则这种测试方法称为黑盒测试。

白盒测试是一种从程序员角度出发的测试，它基于程序的内部结构，检查内部操作是否按规定执行。白盒测试又称为结构测试，即根据被测试程序的内部结构设计测试用例，测试者需要预先了解被测程序的内部结构。

3. 从软件测试的策略和过程的角度分类

从软件测试的策略和过程的角度，软件测试可分为单元测试（unit testing）、集成测试（integration testing）、确认测试（validation testing）、系统测试（system testing）和验收测试（verification testing）。

单元测试是针对每个单元的测试，是软件测试的最小单位，用于确保每个模块能正常工作。单元测试主要采用白盒测试方法，用以发现程序内部结构错误。

集成测试是对已测试过的模块进行组装之后的测试活动，进行集成测试的目的主要在于检验与软件设计相关的程序结构问题。在集成测试过程中，测试人员结合黑盒测试和白盒测试方法，来验证多个单元模块集成到一起后是否能够协调工作。

确认测试是检验所开发的软件能否满足所有功能和性能需求的最后手段，通常采用黑盒测试方法。

系统测试的主要任务是检测被测软件与系统的其他部分的协调性，通常采用黑盒测试方法。

验收测试是软件产品质量的最后一关。这一环节，测试主要从用户的角度着手，其参与者主要是用户和少量的程序开发人员，通常采用黑盒测试方法[5]。

22　静态测试和动态测试

根据程序是否运行可以把软件测试方法分为静态测试和动态测试两大类。静态测试是指被测程序不被运行，而通过其他手段进行检测的测试方法。动态测试则是指通过运行和使用被测程序，发现软件故障，以达到检测目的的测试方法。以汽车检查过程为例，打开前盖检查、检查车身是否刮伤属于静态测试，而上路行驶、检查加速性能属性动态测试。

221　静态测试

静态测试方法的主要特征即为被测程序不被真正运行。静态测试方法包括代码检查、静态结构分析、代码质量度量等。它可以由人工进行，也可以借助软件工具自动进行。通常在静态测试阶段进行以下一些测试活动：

（1）检查算法的逻辑正确性，如检查算法是否实现了所要求的功能。

（2）检查模块接口的正确性，如检查参数个数、参数类型、返回值类型的正确性。

（3）检查调用其他模块接口的正确性，如检查实参个数、实参类型、返回值类型是否

与模块接口匹配,被调用模块是否出现异常。

(4) 检查表达式、语句的正确性,如检查容易产生歧义的表达式或运算符优先级。

(5) 检查常量或全局变量使用的正确性。

(6) 检查输入参数是否有合法性检查。

(7) 检查代码执行效率是否可以优化。

(8) 检查编程风格的规范性,如检查变量命名、代码、注释是否规范。

静态分析是编译程序所不能替代的,编译系统虽然能发现某些程序错误,但这些错误远非软件中存在的大部分错误,如程序的功能性错误。目前,已经开发了一些静态分析系统作为软件静态测试的工具,静态分析已被当作一种自动化的代码校验方法。

222 动态测试

动态测试方法是通过输入有效的测试用例,真正运行被测程序,进行执行跟踪、时间分析以及测试覆盖等方面的测试,并对输入与输出对应关系进行分析,以达到检测的目的。动态测试方法的基本步骤如下:

① 选取程序输入定义域的有效值,或选取定义域外的无效值。

② 决定已选输入值的预期结构。

③ 用选取输入值执行程序。

④ 比较执行结果和预期结果,不吻合则说明程序有错。

动态测试,又可分为基于程序内部结构的白盒测试和基于程序外部功能的黑盒测试。不同测试方法各自的目标和侧重点不一样,在实际工作中要将静态测试和动态测试结合起来,以达到更加好的测试效果。

23 黑盒测试方法

231 黑盒测试方法概述

黑盒测试是一种从用户观点出发的测试,主要以软件规格说明书为依据,对程序功能和程序接口进行的测试。黑盒测试的基本观点是:任何程序都可以看作是从输入定义域映射到输出值域的函数过程,被测程序被认为是一个打不开的黑盒子,黑盒中的内容完全不知道,只明确该黑盒的作用。黑盒测试是软件功能测试的重要手段,它主要的依据是程序规格说明,不涉及其内部结构和特性,只依靠被测程序输入和输出之间的关系设计测试用例。很明显,如果规格说明有误,黑盒测试方法是发现不了的。黑盒测试方法着重测试软件的功能需求,是在程序接口上进行的测试,主要是为了发现以下错误:

(1) 是否有功能错误,是否有功能遗漏。

(2) 是否能够正确地接收输入数据并产生正确的输出结果。

(3) 是否有数据结构错误或外部信息访问错误。

(4) 是否有程序初始化和终止方面的错误。

黑盒测试有两个显著的特点：一是黑盒测试不用考虑软件的具体实现过程，当软件实现的具体过程发生变化时，只要软件接口不变，测试用例仍然可以使用；二是黑盒测试用例的设计可以和软件实现同时进行，这样能够减少总的软件开发时间。黑盒测试不仅能够找到大多数其他测试方法无法发现的错误，而且对于某些无法得到源程序的软件只能选择黑盒测试。黑盒测试的具体技术方法主要包括等价类划分法、边界值分析法、决策表法、因果图法等。

2.3.2 等价类划分法

1. 等价类划分法概述

等价类划分法是一种常用的黑盒测试用例设计方法，它将不能穷举的测试集进行合理分类，从而保证设计出来的有穷测试用例集具有完整性和代表性。

等价类划分法是把所有可能的输入数据(即程序输入的定义域)划分为若干部分(子集)，并从每一个子集中选取少量具有代表性的数据生成测试用例。所谓等价类(equivalence class)是指输入定义域的某个子集，所有等价类的并集就是整个输入定义域。在等价类中，各个输入数据具有等价特性(即对测试程序错误的效果是等价的)。因此，测试某个等价类的代表值就等价于对这一类中其他值的测试。也就是说，如果用等价类中的一个例子作为测试数据进行测试不能发现软件中的故障，那么使用等价类中的其他例子进行测试也不可能发现故障。使用等价类划分法的目的是既希望进行完备的测试，同时又希望避免冗余。

软件不能只接收合理有效的数据，还应经受意外的考验，即接受无效的或不合理的数据，这样获得的软件才能具有较高的可靠性。因此，在进行等价类划分时，应考虑以下两种不同的情况。

1) 有效等价类

有效等价类是指对软件规格说明而言，有意义、合理的输入数据所构成的集合。利用有效等价类，可以检验程序是否实现了规格说明预先规定的功能和性能。在具体问题中，有效等价类可以是一个，也可以是多个。

2) 无效等价类

无效等价类是指对软件规格说明而言，是不合理或无意义的输入数据所构成的集合。利用无效等价类可以检验程序异常情况的处理。对于具体的问题，无效等价类至少应有一个，也可能有多个。

等价类划分法测试的实现可分为两个步骤：一是在分析需求规格说明的基础上划分等价类，然后列出等价类表。等价类表应包含“输入条件”、“有效等价类”和“无效等价类”三方面内容；二是根据已列出的等价类表确定测试用例，确定测试用例步骤：

① 为每个等价类确定一个唯一的编号。

② 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这个过程，直至所有的有效等价类均被测试用例所覆盖。

③ 设计一个新的测试用例，使其仅覆盖一个无效等价类，重复这个过程，直至所有的

无效等价类均被测试用例所覆盖。

例 2-1　以三角形问题为例，要求输入三个正整数，分别作为三角形的三条边，输入值取值范围在 1～100 之间。分析上述的输入条件，可得出相关的等价类表（包括有效等价类和无效等价类），如表 2-1 所示。

表 2-1　三角形问题的等价类

输入条件	等价类编号	有效等价类	等价类编号	无效等价类
三个数	1	三个数	4	只输入一个数
			5	只输入两个数
			6	输入多于三个数
正整数	2	三边为正整数	7	一边为非正整数
			8	两边为非正整数
			9	三边为非正整数
取值范围在 1～100 之间	3	$1\leqslant a\leqslant 100$ $1\leqslant b\leqslant 100$ $1\leqslant c\leqslant 100$	10	一边为 0
			11	两边为 0
			12	三边为 0
			13	一边小于 0
			14	两边小于 0
			15	三边小于 0
			16	一边大于 100
			17	两边大于 100
			18	三边大于 100

2. 常见等价类划分形式

针对是否对无效数据进行测试，可以将等价类测试分为标准等价类测试和健壮等价类测试。

1）标准等价类测试

标准等价类测试不考虑无效输入值，测试用例使用每个等价类中的一个值。通常，标准等价类测试用例的数量和最大等价类中元素的数目相等。

以三角形问题为例，要求输入三个正整数 a、b、c，分别作为三角形的三条边，取值范围在 1～100 之间，判断由三条边构成的三角形类型为等边三角形、等腰三角形、一般三角形或非三角形。在多数情况下，是从输入定义域划分等价类，但对于三角形问题，从输出值域来定义等价类是最简单的划分方法。因此，利用这些信息可以确定下列值域等价类：

$$R1=\{<a,\ b,\ c>:\text{边为 } a、b、c \text{ 的等边三角形}\}$$

R2 = {< a, b, c >：边为 a、b、c 的等腰三角形}
R3 = {< a, b, c >：边为 a、b、c 的一般三角形}
R4 = {< a, b, c >：边为 a、b、c 不构成三角形}

4 个标准等价类测试用例如表 2-2 所示。

表 2-2 三角形问题的标准等价类测试用例

测试用例	*a*	*b*	*c*	预期输出
Test Case 1	8	8	8	等边三角形
Test Case 2	8	8	6	等腰三角形
Test Case 3	4	5	6	一般三角形
Test Case 4	1	2	4	不构成三角形

2）健壮等价类测试

健壮等价类测试考虑了无效等价类。对有效输入，测试用例从每个有效等价类中取一个值（和标准等价类测试类似）；对无效输入，一个测试用例有一个无效值，其他值均取有效值。

健壮等价类测试存在两个问题：一是需要花费时间定义无效测试用例的期望输出；二是对强类型的语言没有必要考虑无效的输入。

对于三角形问题，考虑三角形三条边 a、b、c 的无效值的健壮等价类测试用例如表 2-3 所示。

表 2-3 三角形问题的健壮等价类测试用例

测试用例	*a*	*b*	*c*	预 期 输 出
Test Case 1	8	8	8	等边三角形
Test Case 2	8	8	6	等腰三角形
Test Case 3	4	5	6	一般三角形
Test Case 4	1	2	4	不构成三角形
Test Case 5	0	5	5	a 值不在允许范围内
Test Case 6	5	0	5	b 值不在允许范围内
Test Case 7	5	5	0	c 值不在允许范围内
Test Case 8	101	5	5	a 值不在允许范围内
Test Case 9	5	101	5	b 值不在允许范围内
Test Case 10	5	5	101	c 值不在允许范围内

2.3.3 边界值分析法

1. 边界值分析法概述

人们从长期的测试工作经验得知，大量的故障往往发生在输入定义域或输出值域的边界上，而不是在其内部。边界值分析法（Boundary Value Analysis，BVA）是一种补充等价类划分法的测试用例设计技术，它不是选择等价类的任意元素，而是选择等价类边

界的测试用例,针对各种边界情况设计测试用例,更有可能查出程序错误。

在实际的软件开发过程中,会涉及到大量的边界值条件,例如下面的 Java 程序:

```
int data[]=new int[10];
for (int i=1; i<=10; i++) {
    data[i]=1;
}
```

这段代码的意图是创建包含 10 个元素的数组并为数组中的每一个元素赋初值 1,看似合理,但是,在 Java 语言(大多数编程语言也都类似)中,当一个数组被定义时,其第一个元素所对应的数组下标是 0 而不是 1。这时,如果其他程序员在使用这个数组的时候,数组的第一个元素没有被正常初始化,可能会造成软件的缺陷或者错误的产生。诸如此类问题很常见。

使用边界值分析方法设计测试用例,首先应确定边界情况。通常输入和输出等价类的边界,就是应着重测试的边界情况。应选取正好等于、刚刚大于或刚刚小于边界的值作为边界测试数据,应用边界值分析法设计测试用例可借鉴以下经验:

(1) 若输入条件规定了输入值的范围,则应该选取刚达到这个范围的边界值,以及刚刚超过这个范围边界的值作为测试输入数据。

(2) 若输入条件规定了输入值的个数,则用最大个数、最小个数、比最大个数少 1、比最大个数多 1 的数作为测试数据。

(3) 若输入域或输出域是有序集合,则应选取集合的第一个元素和最后一个元素作为测试用例。

(4) 若程序中使用了一个内部数据结构,则应当选择这个内部数据结构的边界值作为测试用例。

(5) 根据规格说明的每一个输入条件分别设计边界测试用例。

2. 边界条件与次边界条件

上面讨论的边界条件比较容易发现,它们在软件规格说明中或者有定义,或者可以在使用软件的过程中确定。然而,在测试用例设计过程中,某些边界值条件在软件内部,是不需要呈现给用户的,或者说用户很难注意到这些问题,但这些边界条件确实属于检验范畴内,称为次边界值条件,主要有以下几种。

1) 数值的次边界值检验

计算机是基于二进制进行工作的,因此,任何数值运算都有一定的范围限制,都由相应的 2 的幂次方表示,常用数值运算范围如表 2-4 所示。

表 2-4 计算机数值运算的范围

术语	范围	术语	范围
位(b)	0 或 1	千(K)	1024
字节(B)	0～255	兆(M)	1 048 576
字(word)	0～65 535 或 0～4 294 967 295	吉(G)	1 073 741 824

例如,对字节进行检验,边界值条件可以设置为 254、255 和 256。

2) 字符的次边界值检验

在字符的编码方式中,ASCII 和 Unicode 是比较常见的编码方式,表 2-5 中列出了一些字符的 ASCII 码值。

表 2-5 字符的 ASCII 码值对应表

字　符	ASCII 码值	字　符	ASCII 码值
null	0	B	66
space	32	Y	89
/	47	Z	90
0	48	[	91
1	49	‘	96
2	50	a	97
9	57	b	98
:	58	y	121
@	64	z	122
A	65	{	123

例如,如果测试的文本框只允许用户输入字符 A～Z 或 a～z,ASCII 表中的值,则在设计边界测试用例时,应该在非法区间中包含 ASCII 表中这些字符前后的值,如@、[、’、{。

3) 其他应用相关的次边界值检验

许多边界值与软件开发所使用的平台、系统、工具等相关。例如,一个人口分布统计系统,在录入年份时,页面年份的检查范围是:[1900,2006]。该范围以内的没问题,范围外的给出提示。但如果数据库采用的 MySQL,年份字段类型为 Year,那么该检查范围就有问题,因为 Year 的范围为[1901,2155],当输入年份 1900 保存,数据库中的数据为 0000。因此页面年份的检查范围应为[1900,2006],而次边界条件为:[1901,2155]。

其他的边界值检验包括默认值、空值、空格、未输入值、0、无效数据、不正确数据和干扰数据等。在实际的测试用例设计中,需要将基本的软件设计要求和程序定义的要求结合起来,即结合基本边界值条件和次边界值条件来设计有效的测试用例。

3. 边界值分析法测试用例

以三角形问题为例,要求输入三个整数 a、b、c,分别作为三角形的三条边,取值范围在 1～100 之间,判断由三条边构成的三角形类型为等边三角形、等腰三角形、一般三角形或非三角形。表 2-6 给出了边界值分析测试用例。

2.3.4 决策表法

1. 决策表法概述

在所有的黑盒测试方法中,基于决策表的测试是最为严格、最具有逻辑性的测试方

表 2-6 边界值分析测试用例

测试用例	*a*	*b*	*c*	预期输出
Test Case 1	50	50	1	等腰三角形
Test Case 2	50	50	2	等腰三角形
Test Case 3	100	100	100	等边三角形
Test Case 4	50	50	99	等腰三角形
Test Case 5	50	50	100	非三角形
Test Case 6	50	1	50	等腰三角形
Test Case 7	50	2	50	等腰三角形
Test Case 8	50	99	50	等腰三角形
Test Case 9	50	100	50	非三角形
Test Case 10	1	50	50	等腰三角形
Test Case 11	2	50	50	等腰三角形
Test Case 12	99	50	50	等腰三角形
Test Case 13	100	50	50	非三角形

法。在一些数据处理问题中，某些操作的实施依赖于多个逻辑条件的组合，即针对不同逻辑条件的组合值，分别执行不同的操作，决策表很适合于处理这类问题。决策表通常由四个部分组成(如图 2-1 所示)。

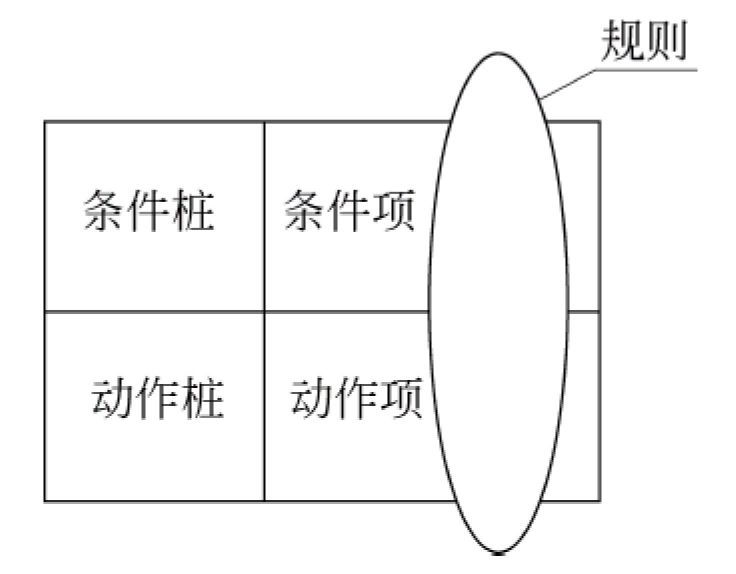


图 2-1 决策表的组成

(1) 条件桩：列出了问题的所有条件，通常认为列出的条件的先后次序无关紧要。

(2) 动作桩：列出了问题规定的可能采取的操作，这些操作的排列顺序没有约束。

(3) 条件项：针对条件桩给出的条件列出所有可能的取值。

(4) 动作项：与条件项紧密相关，指出在条件项的各组取值情况下应采取的动作。

任何一个条件组合的取值及其相应要执行的操作称为一条规则，在决策表中贯穿条件项和动作项的一列就是一条规则。因此，决策表中列出多少组条件取值，也就有多少条规则，即条件项和动作项有多少列。实际使用决策表时，常常先将其简化，简化是以合并相似规则为目标的。若表中有两条或多条规则具有相同的动作，并且在条件项之间存在着极为相似的关系，便可以设法将其合并。根据软件规格说明，构造决策表的步骤如下：

(1) 确定规则的个数。假设有 n 个条件，每个条件有 m 个取值，则共有 m^n 种规则；

(2) 列出所有的条件桩和动作桩。

(3) 填入条件项和动作项，得到初始决策表。

(4) 简化决策表。

以如下问题为例给出构造决策表的具体过程：

例 2-2 如果某产品销售好且库存低，则增加该产品的生产；如果该产品销售好，但库存量不低，则维持生产；若该产品销售不好，但库存量低，则也维持生产；若该产品销售

不好，且库存量不低，则停止生产。

解法如下：

（1）确定规则的个数：对于本题有两个条件（销售情况和库存情况），每个条件可以有两个取值（销售好、不好，库存低、不低），故有 $2^2=4$ 种规则。

（2）列出所有的条件桩和动作桩。

（3）填入条件项和动作项，得到初始决策表，如表 2-7 所示。

表 2-7　产品销售问题的决策表

选项＼规则		1	2	3	4
条件	c_1：销售是否好	T	T	F	F
	c_2：库存是否低	T	F	T	F
动作	a_1：增加生产	√			
	a_2：维持生产		√	√	
	a_3：停止生产				√

2. 决策表法的应用

决策表最突出的优点是，它能够将复杂的问题按照各种可能的情况一一列举出来，简明并避免遗漏。因此，利用决策表能够设计出完整的测试用例集合。使用决策表设计测试用例，可以把条件理解为输入，把动作理解为输出。

以三角形问题为例，要求输入三个整数 a、b、c，分别作为三角形的三条边，取值范围在 1～100 之间，判断由三条边构成的三角形类型为等边三角形、等腰三角形、一般三角形或非三角形。

分析如下：

（1）确定规则的个数。例如，三角形问题的决策表有 4 个条件（三边是否构成三角形、三边是否两两相等），每个条件可以取两个值（真和假），所以应有 $2^4=16$ 种规则。

（2）列出所有条件桩和动作桩（五种结果：非三角形、一般三角形、等腰三角形、等边三角形和不可能的取值）。

（3）填写条件项。

（4）填写动作项，从而得到初始决策表，如表 2-8 所示。

（5）简化决策表，合并相似规则后得到三角形问题的简化决策表，如表 2-9 所示（合并后的条件项用—表示动作项与其取值无关，称为“无关条件”或“不关心条件”）。

根据决策表 2-9，可设计测试用例，如表 2-10 所示。

2.3.5 因果图法

等价类划分法和边界值分析法都着重考虑输入条件，而没有考虑到输入条件的各种组合情况，也没有考虑到各个输入条件之间的相互制约关系。如果在测试时必须考虑输

表 2-8　三角形问题的初始决策表

选项 \ 规则		1	2	3	4	5	6	7	8
条件	c_1：a、b、c 是否构成一个三角形	F	F	F	F	F	F	F	F
	c_2：$a=b$?	T	T	T	T	F	F	F	F
	c_3：$b=c$?	T	T	F	F	T	T	F	F
	c_4：$a=c$?	T	F	T	F	T	F	T	F
动作	a_1：非三角形	√	√	√	√	√	√	√	√
	a_2：一般三角形								
	a_3：等腰三角形								
	a_4：等边三角形								
	a_5：不可能								

选项 \ 规则		9	10	11	12	13	14	15	16
条件	c_1：a、b、c 是否构成一个三角形	T	T	T	T	T	T	T	T
	c_2：$a=b$?	T	T	T	T	F	F	F	F
	c_3：$b=c$?	T	T	F	F	T	T	F	F
	c_4：$a=c$?	T	F	T	F	T	F	T	F
动作	a_1：非三角形								
	a_2：一般三角形								√
	a_3：等腰三角形				√		√	√	
	a_4：等边三角形	√							
	a_5：不可能		√	√		√			

表 2-9　三角形问题的简化决策表

选项 \ 规则		1～8	9	10	11	12	13	14	15	16
条件	c_1：a、b、c 是否构成一个三角形	F	T	T	T	T	T	T	T	T
	c_2：$a=b$?	—	T	T	T	T	F	F	F	F
	c_3：$b=c$?	—	T	T	F	F	T	T	F	F
	c_4：$a=c$?	—	T	F	T	F	T	F	T	F
动作	a_1：非三角形	√								
	a_2：一般三角形									√
	a_3：等腰三角形					√		√	√	
	a_4：等边三角形		√							
	a_5：不可能			√	√		√			

表 2-10 三角形问题的决策表测试用例

测试用例	a	b	c	预期输出
Test Case 1	4	1	2	非三角形
Test Case 2	1	4	2	非三角形
Test Case 3	1	2	4	非三角形
Test Case 4	3	3	3	等边三角形
Test Case 5	?	?	?	不可能
Test Case 6	?	?	?	不可能
Test Case 7	3	3	4	等腰三角形
Test Case 8	?	?	?	不可能
Test Case 9	4	3	3	等腰三角形
Test Case 10	3	4	3	等腰三角形
Test Case 11	3	4	5	一般三角形

入条件的各种组合,可能的组合数将是一个天文数字,因此必须考虑采用一种适合于多种条件的组合,产生多个动作的形式来进行测试用例的设计,这就需要采用因果图法。因果图法就是一种利用图解法分析输入的各种组合情况,从而设计测试用例的方法,它适用于检查程序输入条件的各种情况的组合。

因果图中使用简单的逻辑符号,以直线连接左右结点,如图 2-2 所示。左结点 c_i 表示输入状态(或称原因),右结点 e_i 表示输出状态(或称结果)。在因果图中使用 4 种符号分别表示 4 种因果关系,c_i 和 e_i 都可取值 0 或 1,0 表示某状态不出现,1 表示某状态出现。

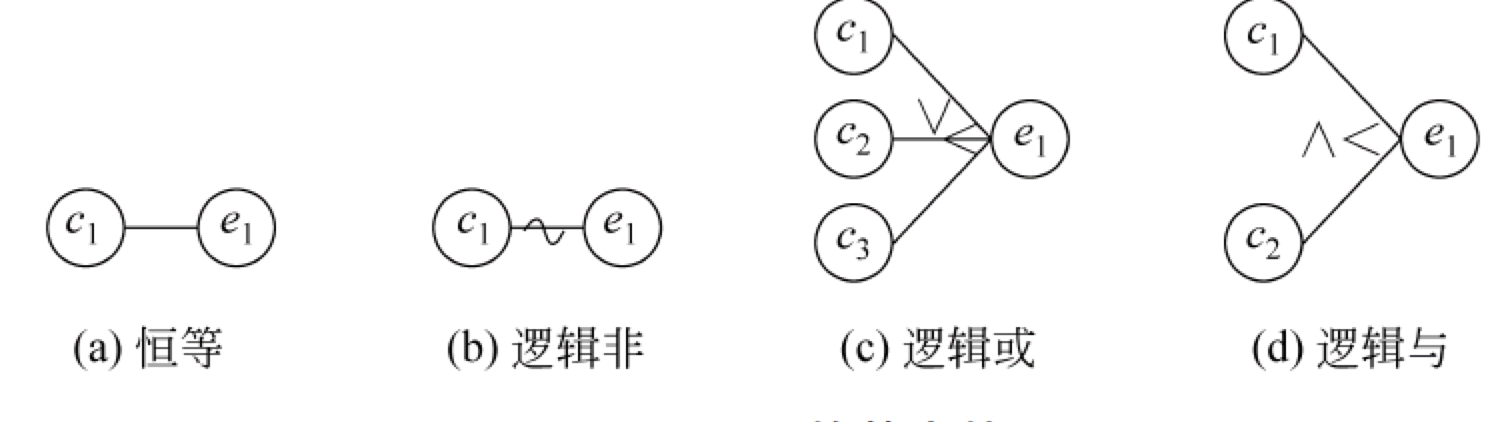


图 2-2 因果图的基本符号

图 2-2(a)中,若 c_1 是 1,则 e_1 也是 1;否则 e_1 为 0。

图 2-2(b)中,若 c_1 是 1,则 e_1 是 0;否则 e_1 为 1。

图 2-2(c)中,若 c_1 或 c_2 或 c_3 是 1,则 e_1 是 1;否则 e_1 为 0。“或”可以有任意多个输入。

图 2-2(d)中,若 c_1 与 c_2 都是 1,则 e_1 是 1;否则 e_1 为 0。“与”可以有任意多个输入。

在实际问题中,输入状态相互之间还可能存在某些依赖关系,称之为“约束”。比如,某些输入条件本身不可能同时出现。输出状态之间也往往存在约束。在因果图中,用特定的符号标明这些约束(如图 2-3 所示)。

图 2-3 中对输入条件的约束有如下 4 种:

图 2-3(a)表示 E 约束(异)。a 和 b 中最多有一个可能为 1,即 a 和 b 不能同时为 1。

图 2-3(b)表示 I 约束(或)。a、b 和 c 中至少有一个必须是 1,即 a、b 和 c 不能同时为 0。

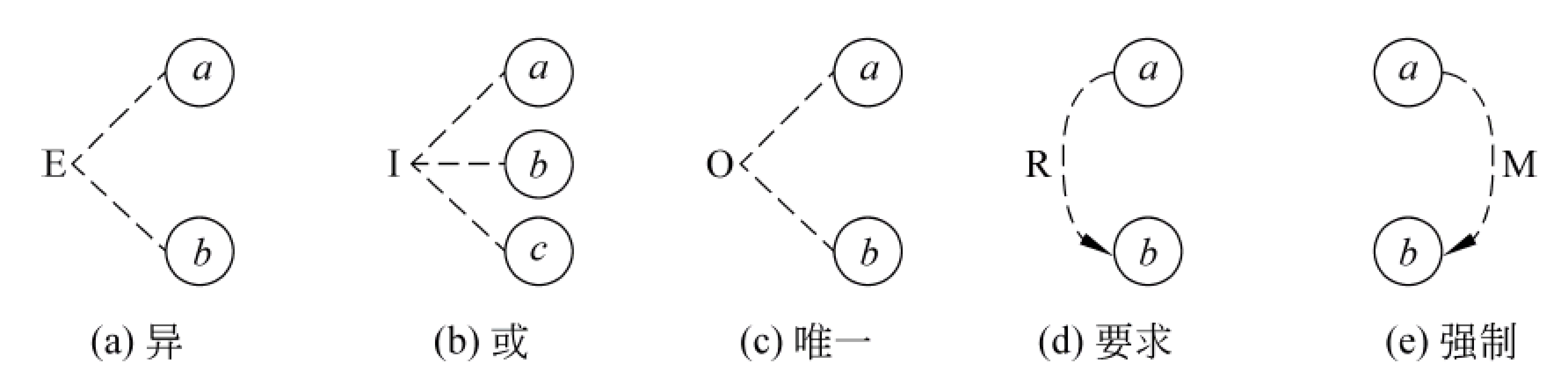


图 2-3 约束符号

图 2-3(c)表示 O 约束(唯一)。a 和 b 必须有一个且仅有 1 个为 1。

图 2-3(d)表示 R 约束(要求)。a 是 1 时,b 必须是 1,即当 a 是 1 时,b 不能是 0。

对输出条件的约束只有 M 约束。

图 2-3(e)表示 M 约束(强制)。若结果 a 是 1,则结果 b 强制为 0。

因果图方法最终生成决策表。利用因果图导出测试用例需要经过以下几个步骤:

① 分析软件规格说明书中哪些是原因,哪些是结果。原因常常是输入条件或输入条件的等价类,结果则是输出条件。

② 分析软件规格说明书中语义的内容,找出原因与结果之间,原因与原因之间的对应关系,并将其表示成连接各个原因与各个结果的"因果图"。

③ 由于语法或环境的限制,有些原因与原因之间,原因与结果之间的组合情况不可能出现。为表明这些特定的情况,在因果图上使用一些记号标明约束或限制条件。

④ 把因果图转换成决策表。

⑤ 根据决策表中每一列设计测试用例。

下面以一个简单的例子,说明因果图方法的步骤。

例 2-3 某软件规格说明要求:第一个字符必须是+或 *,第二个字符必须是一个数字,在此情况下进行文件的修改。如果第一个字符不是+或 *,则给出信息 N;如果第二个字符不是数字,则给出信息 M。

在分析以上的要求以后,可以明确地把原因和结果分开如下。

原因:

- c_1——第一个字符是+。
- c_2——第一个字符是 *。
- c_3——第二个字符是一数字。

结果:

- e_1——给出信息 N。
- e_2——修改文件。
- e_3——给出信息 M。

将原因和结果用上述的逻辑符号联接起来,可以得到如图 2-4 所示的因果图。图中左边表示原因,右边表示结果,编号为 10 的中间结点是导出结果的进一步原因(即原因 c_1 和 c_2 通过逻辑或连接得到的中间原因)。

考虑到原因 c_1 和 c_2 不可能同时为 1,即第一个字符不可能既是+又是 *,在因果图

上可对其施加 E 约束，这样便得到了具有约束的因果图，如图 2-5 所示。根据因果图可以建立如表 2-11 所示的决策表(其中条件项中 1 代表相应原因满足，0 代表不满足)。

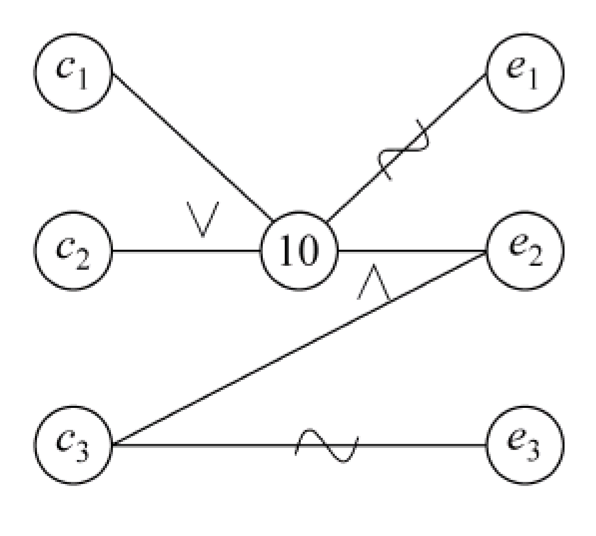


图 2-4 因果图表示

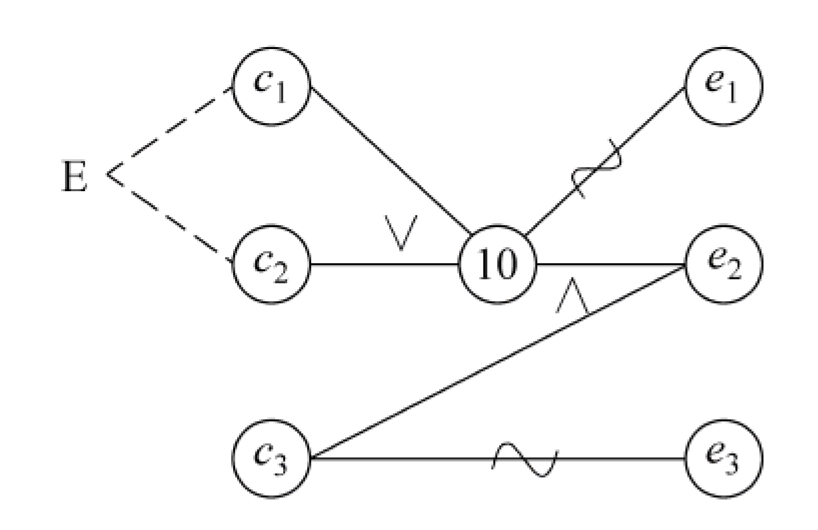


图 2-5 具有 E 约束的因果图表示

表 2-11 根据因果图建立的决策表

选项 \ 规则		1	2	3	4	5	6	7	8
条件	c_1	1	1	1	1	0	0	0	0
	c_2	1	1	0	0	1	1	0	0
	c_3	1	0	1	0	1	0	1	0
	10			1	1	1	1	0	0
动作	e_1							√	√
	e_2			√		√			
	e_3				√		√		√
	不可能	√	√						

注意，表中 8 种情况的最左面两列，原因 c_1 和 c_2 同时为 1，这是不可能的，故应排除这两种情况。根据该表，可设计出 6 个测试用例。

以上只是因果图应用的一个简单例子，在较为复杂的问题中，因果图方法常常十分有效，它能有效地帮助检查输入条件组合，设计出非冗余、高效的测试用例。当然，如果开发项目在设计阶段就采用了决策表，就不必再画因果图，可以直接利用决策表设计测试用例。

2.3.6 各种黑盒测试方法的选择

为了最大程度地发现存在的软件缺陷，在测试实施之前，测试工程师必须确定将要采用的黑盒测试策略和方法，并以此为依据制定详细的测试方案，一个好的测试策略和测试方法将给整个测试工作带来事半功倍的效果。如何才能确定好的黑盒测试策略和测试方法呢？通常，在确定黑盒测试方法时，应遵循以下原则：

(1) 根据程序的重要性和一旦发生故障将造成的损失程度来确定测试等级和测试重点。

(2) 认真选择测试策略，以便能尽可能少地使用测试用例，发现尽可能多的程序错

误。经过一次完整的软件测试，如果程序中遗漏的问题过多并且严重，则表明该次测试是不足的，测试不足意味着让用户承担隐藏错误带来的危险，但测试过度又会带来资源的浪费。因此，测试需要找到一个平衡点。

以下是各种黑盒测试方法选择的综合策略，可在实际应用过程中参考：

（1）首先进行等价类划分，包括输入条件和输出条件的等价划分，将无限测试变成有限测试，这是减少工作量和提高测试效率的最有效方法。

（2）在任何情况下都必须使用边界值分析方法。经验表明用这种方法设计出的测试用例发现程序错误的能力最强。

（3）如果程序的功能说明中含有输入条件的组合情况，则应在一开始就选用因果图法。

23.7　黑盒测试的优缺点

黑盒测试是一种确认技术，目的是确认“设计的系统是否正确”。黑盒测试是以用户的观点，考虑输入数据与输出数据的对应关系，而不考虑内部结构及工作情况；若外部特性本身存在问题或规格说明的规定有误，则应用黑盒测试方法是不能发现问题的。

黑盒测试的优点如下：

（1）适用于各个测试阶段。

（2）从产品功能角度进行测试。

（3）容易入手生成测试数据。

黑盒测试的缺点如下：

（1）某些代码得不到测试。

（2）如果规格说明有误，无法发现。

（3）不易进行充分测试。

24　白盒测试方法

白盒测试是基于被测程序的源代码，而不是软件规格说明的测试活动。白盒测试按照程序内部的结构测试程序，检验程序中的每条通路是否都能按预定要求正确工作，而不考虑它的功能。白盒测试的主要方法有逻辑覆盖、路径分析测试等。

24.1　逻辑覆盖测试

逻辑覆盖测试基于程序的逻辑结构设计相应的测试用例，要求测试人员深入了解被测程序的逻辑结构特点，完全掌握源代码的流程。根据不同的测试要求，逻辑覆盖测试可以分为语句覆盖、判断覆盖、条件覆盖、判断/条件覆盖、条件组合覆盖和路径覆盖。

下面是一段简单的 Java 语言程序，作为公共程序段来讨论五种覆盖测试的各自特点。

程序 2-1：

```
1    if (x> 100 && y> 500)
2            result= result+ 100;
3    if (x> = 1000|| z> 5000)
4            result= result- 50;
```

其程序控制流图如图 2-6 所示。

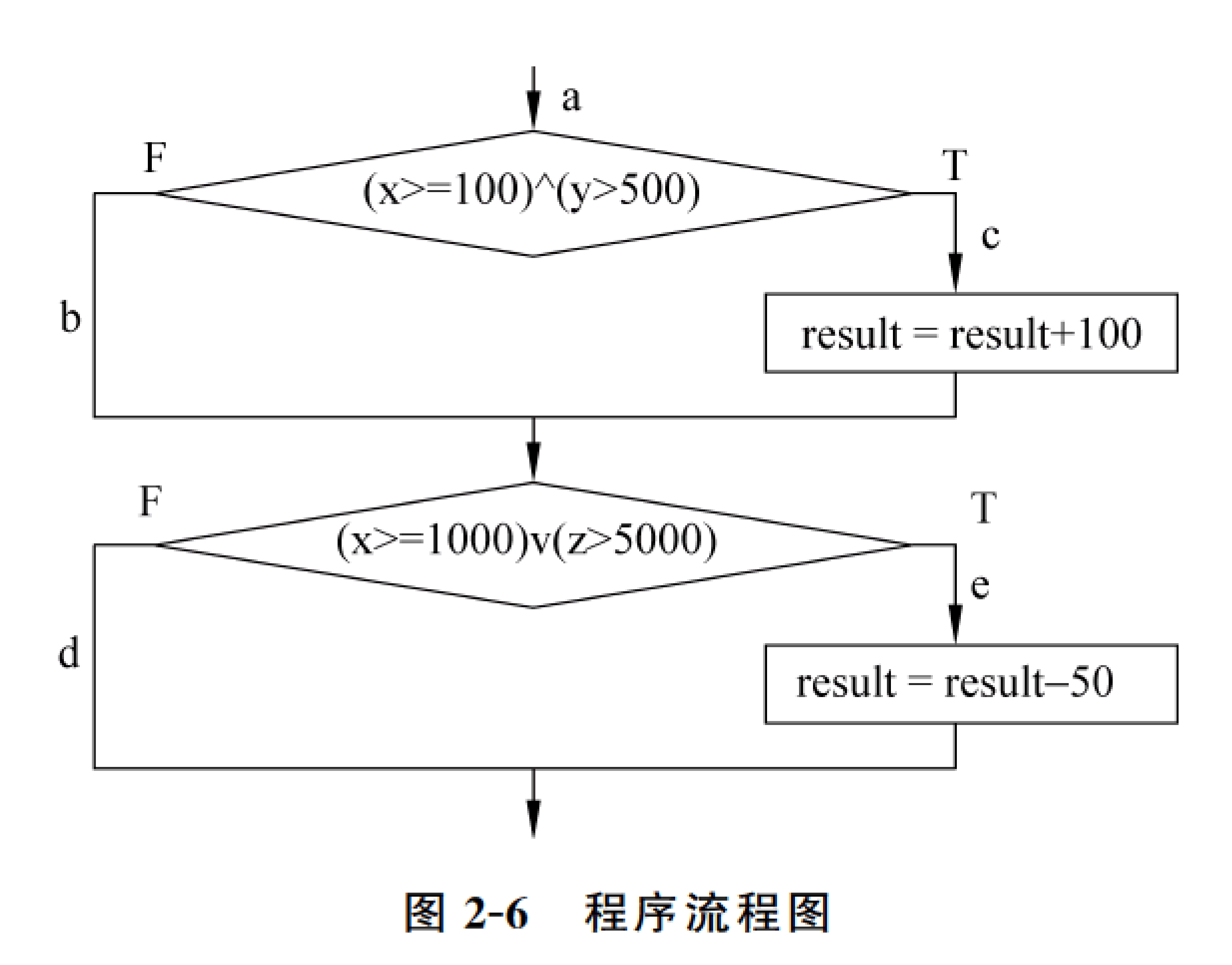


图 2-6　程序流程图

1. 语句覆盖

语句覆盖(statement coverage)要求设计若干个测试用例,运行被测程序,使得每个可执行语句至少被执行一次。在保证完成要求的情况下,测试用例的数目越少越好。

以下是针对程序 2-1 设计的两个测试用例,称为测试用例组 1。

```
Test Case 1: x= 1000, y= 600, z= 6000
Test Case 2: x= 200, y= 600, z= 5000
```

如表 2-12 所示,采用 Test Case 1 作为测试用例,则程序按路径 a→c→e 顺序执行,程序中的 4 个语句都被执行一次,符合语句覆盖的要求。采用 Test Case 2 作为测试用例,则程序按路径 a→c→d 顺序执行,程序中的语句 4 没有执行到,所以没有达到语句覆盖的要求。

表 2-12　测试用例组 1

测试用例	x，y，z	(x>100)&&(y>500)	(x>=1000)‖(z>5000)	执行路径
Test Case 1	1000，600，6000	True	True	ace
Test Case 2	200，600，5000	True	False	acd

从表面上看,语句覆盖用例测试了程序中的每一个语句行,似乎能够比较全面地检验被测对程序,但实际上语句覆盖测试是最弱的逻辑覆盖方法。例如,若第一个判断的逻辑运算符 && 被误写成 ‖,或者第二个判断的逻辑运算符 ‖ 被误写成 &&,这时如果采用 Test Case 1 测试用例是检验不出程序中的判断逻辑错误的。又如果将语

句 3 误写成 if (x>=1500 ‖ z>5000),Test Case 1 同样无法发现错误之处。

因此,语句覆盖测试只是表面上覆盖程序流程,没有针对源程序各个语句间的内在联系,难以排除被测程序中存在的潜在故障。

2. 判断覆盖

判断覆盖(branch coverage)要求设计若干个测试用例,运行被测程序,使得程序中每个判断条件的真值分支和假值分支至少被执行一遍。在保证完成要求的情况下,测试用例的数目越少越好。判断覆盖又称为分支覆盖。

仍以程序 2-1 为例,设计测试用例组 2:

```
Test Case 1: x=1000, y=600, z=6000
Test Case 3: x=50, y=600, z=2000
```

如表 2-13 所示,采用 Test Case 1 作为测试用例,程序按路径 a→c→e 顺序执行;采用 Test Case 3 作为测试用例,程序按路径 a→b→d 顺序执行。所以采用这一组测试用例,程序 2-1 的 4 个判断分支 b→c→d→e 都被覆盖到了。

表 2-13 测试用例组 2

测试用例	x, y, z	(x>100)&&(y>500)	(x≥1000) ‖ (z>5000)	执行路径
Test Case 1	1000, 600, 6000	True	True	ace
Test Case 3	50, 600, 2000	False	False	abd

或者设计测试用例组 3:

```
Test Case 4: x=200, y=600, z=2000
Test Case 5: x=1000, y=200, z=6000
```

如表 2-14 所示,采用 Test Case 4 作为测试用例,程序按路径 a→c→d 顺序执行;采用 Test Case 5 作为测试用例,程序按路径 a→b→e 顺序执行。同样可以满足判断覆盖。

表 2-14 测试用例组 3

测试用例	x, y, z	(x>100)&&(y>500)	(x≥1000) ‖ (z>5000)	执行路径
Test Case 4	200, 600, 2000	True	False	acd
Test Case 5	1000, 200, 6000	False	True	abe

实际上,上述两组测试用例不仅达到了判断覆盖要求,也同时满足了语句覆盖要求,从这点上看判断覆盖测试要强于语句覆盖测试。但是,判断覆盖测试依然不够完善。例如,如果将第二个判断条件(x>=1000) ‖ (z>5000)中的 z>5000 错误定义成(x>=1000) ‖ (z>8000),由于判断条件中的两个判断是“或”关系,其中一个判断式错误是不影响结果的,所以这两组测试用例发现不了该问题。因此,应该用具有更强逻辑覆盖能力的覆盖测试方法来测试这种内部判断条件。

3. 条件覆盖

条件覆盖(condition coverage)要求设计若干个测试用例,执行被测程序,使得程序中每个判断条件中的每个判断式的真值和假值至少被执行一遍。

设计测试用例组 4:

```
Test Case 1: x=1000, y=600, z=6000
Test Case 3: x=50, y=600, z=2000
Test Case 6: x=1000, y=200, z=2000
```

如表 2-15 所示,测试用例组中的 3 个测试用例覆盖了 4 个内部判断式的 8 种真假值情况。此外,这组测试用例也同时实现了判断覆盖,即覆盖了所有 4 个判断分支 b、c、d、e。但是否可以说实现了条件覆盖就必然实现了判断覆盖呢?下面通过测试用例组 5 对这个问题进行分析。

表 2-15 测试用例组 4

测试用例	x, y, z	(x>100)	(y>500)	(x≥1000)	(z>5000)	执行路径
Test Case 1	1000, 600, 6000	True	True	True	True	ace
Test Case 3	50, 600, 2000	False	True	False	False	abd
Test Case 6	1000, 200, 2000	True	False	True	False	abe

设计测试用例组 5:

```
Test Case 7: x=50, y=600, z=6000
Test Case 8: x=2000, y=200, z=1000
```

如表 2-16 和表 2-17 所示,其中表 2-16 表示每个判断条件的每个判断式的真值和假值,表 2-17 表示每个判断条件的真值和假值。测试用例组 5 中的两个测试用例虽然覆盖了 4 个内部判断式的 8 种真假值情况,但这两个测试用例的执行路径均为 a→b→e,仅覆盖了判断条件的 4 个真假分支中的两个。因此,满足了条件覆盖并不一定就满足判断覆盖,为解决这一矛盾,需要设计一种能兼顾判断覆盖和条件覆盖的覆盖测试方法,即判断/条件覆盖测试。

表 2-16 测试用例组 5——每个判断条件的每个判断式的真值和假值

测试用例	x, y, z	(x>100)	(y>500)	(x≥1000)	(z>5000)	执行路径
Test Case 7	50, 600, 6000	False	True	False	True	abe
Test Case 8	2000, 200, 1000	True	False	True	False	abe

表 2-17 测试用例组 6——每个判断条件的真值和假值

测试用例	x, y, z	(x>100)&&(y>500)	(x≥1000)‖(z>5000)	执行路径
Test Case 7	50, 600, 6000	False	True	abe
Test Case 8	2000, 200, 1000	False	True	abe

4. 判断/条件覆盖

判断/条件覆盖要求设计若干个测试用例,执行被测程序,使得程序中每个判断条件的真假值分支至少被执行一遍,并且每个判断条件的内部判断式的真假值分支也要被执行一遍。

设计测试用例组 6:

```
Test Case 1: x=1000, y=600, z=6000
Test Case 9: x=50, y=200, z=2000
```

如表 2-18 和表 2-19 所示,其中表 2-18 表示每个判断条件的每个判断式的真值和假值,表 2-19 表示每个判断条件的真值和假值。从表中可以看出测试用例组 6 同时满足了判断覆盖和条件覆盖。然而,判断/条件覆盖没有对每个判断条件的内部判断式的所有真假值组合进行测试。条件组合判断是必要的,因为条件判断语句中的“与”和“或”(即 && 和 ‖),会使内部判断式之间产生抑制作用。例如,C=A && B 中,如果 A 为假值,无论 B 为真值或假值,C 都为假值,因此 B 的正确与否没有得到测试。同样,C=A ‖ B 中,如果 A 为真值,那么无论 B 为真值或假值,C 都为真值,B 的正确与否也没有得到测试。

表 2-18 测试用例组 6——每个判断条件的每个判断式的真值和假值

测试用例	x, y, z	(x>100)	(y>500)	(x≥1000)	(z>5000)	执行路径
Test Case 1	2000, 600, 6000	True	True	True	True	ace
Test Case 9	50, 200, 2000	False	False	False	False	abd

表 2-19 测试用例组 6——每个判断条件的真值和假值

测试用例	x, y, z	(x>100)&&(y>500)	(x≥1000)‖(z>5000)	执行路径
Test Case 1	2000, 600, 6000	True	True	ace
Test Case 9	50, 200, 2000	False	False	abd

5. 条件组合覆盖

条件组合覆盖要求设计若干个测试用例,执行被测程序,使得程序中每个判断条件的内部判断式的各种真假值组合可能都至少被执行一遍。因此,满足条件组合覆盖的测试用例一定满足判断覆盖、条件覆盖和判断/条件覆盖。

设计测试用例组 7:

```
Test Case 1: x=1000, y=600, z=6000
Test Case 7: x=50, y=600, z=6000
Test Case 8: x=2000, y=200, z=1000
Test Case 9: x=50, y=200, z=2000
```

如表 2-20 所示,该表表示每个判断条件的每个判断式的真值和假值的组合(即每个判断条件的两个判断式具有 4 种真假值组合)。测试用例组 7 固然覆盖了所有判断式条

件组合,同时也满足了判断覆盖和条件覆盖,然而其没有执行程序所有可能的路径(只执行了路径 a→c→e、a→b→e 和 a→b→d,而未执行路径 a→c→d)。而路径能否被全面覆盖对软件测试非常重要,因为要尽可能地发现软件中存在的缺陷,就必须保证程序沿每条路径都能顺利执行,能够达到这样要求的是路径覆盖测试。

表 2-20 测试用例组 7

测试用例	x，y，z	(x>100)	(y>500)	(x≥1000)	(z>5000)	执行路径
Test Case 1	1000，600，6000	True	True	True	True	ace
Test Case 7	50，600，6000	False	True	False	True	abe
Test Case 8	2000，200，1000	True	False	True	False	abe
Test Case 9	50，200，2000	False	False	False	False	abd

6. 路径覆盖

路径覆盖(path coverage)要求设计若干测试用例,执行被测程序,使得程序中所有可能的路径都能被覆盖到。

程序 2-1 共有 4 条可能路径(即 a→c→e、a→b→e、a→b→d 和 a→c→d),设计测试用例组 8:

```
Test Case 1: x=1000, y=600, z=6000
Test Case 3: x=50, y=600, z=2000
Test Case 4: x=200, y=600, z=2000
Test Case 8: x=2000, y=200, z=1000
```

如表 2-21 所示,该表表示每个判断条件的真值和假值。测试用例组 8 可以达到路径覆盖。

表 2-21 测试用例组 8

测试用例	x，y，z	(x>100)&&(y>500)	(x≥1000)‖(z>5000)	执行路径
Test Case 1	2000，600，6000	True	True	ace
Test Case 3	50，600，2000	False	False	abd
Test Case 4	200，600，2000	True	False	acd
Test Case 8	2000，200，1000	False	True	abe

需要注意的是,程序 2-1 非常简短,只有 4 条可能路径。然而在实际测试程序中,其路径数目可能是一个庞大的数字,要对其实现路径覆盖测试通常是难以实现的。所以,在实际操作中,通常尽可能把路径数压缩到一个可承受的范围。例如,假设程序中的循环体只执行一次。

当然,即使对程序做到了路径覆盖测试,也不能保证被测程序不存在问题,其他的软件测试手段也是必要的。事实上,并不存在一种完美的测试方法,测试的目的只能说是尽可能多地查找软件缺陷。

2.4.2 路径分析测试

从 2.4.1 节的讨论中可以看出,对于比较简单的小程序来说,实现路径覆盖是可能

的，但如果程序比较复杂，存在多个判断和多个循环，可能的路径数目将会急剧增长，以致路径覆盖难以实现。实际上可以做到的只是有选择地测试程序中某些有代表性的路径。独立路径选择和 Z 路径覆盖是两种常见的路径覆盖方法。

1. 控制流图

白盒测试是针对软件产品内部逻辑结构进行的测试，测试人员必须对被测软件有深入的理解，这是一项庞大并且复杂的工作。为了突出程序的内部结构，可以采用控制流图(control flow graph)表示程序流程。相对于源代码，控制流图直观易懂，便于测试人员理解程序内部结构。

1）控制流图的特点

控制流图是由结点和控制边组成的，具有以下几个特点：

(1) 具有唯一入口结点，即源结点，表示程序段的开始语句。

(2) 具有唯一出口结点，即汇结点，表示程度段的结束语句。

(3) 结点由带有标号的圆圈表示，表示一个或多个无分支的源程序语句。

(4) 控制边由带箭头的直线或弧表示，代表控制流的方向。

常见的控制流图如图 2-7 所示。

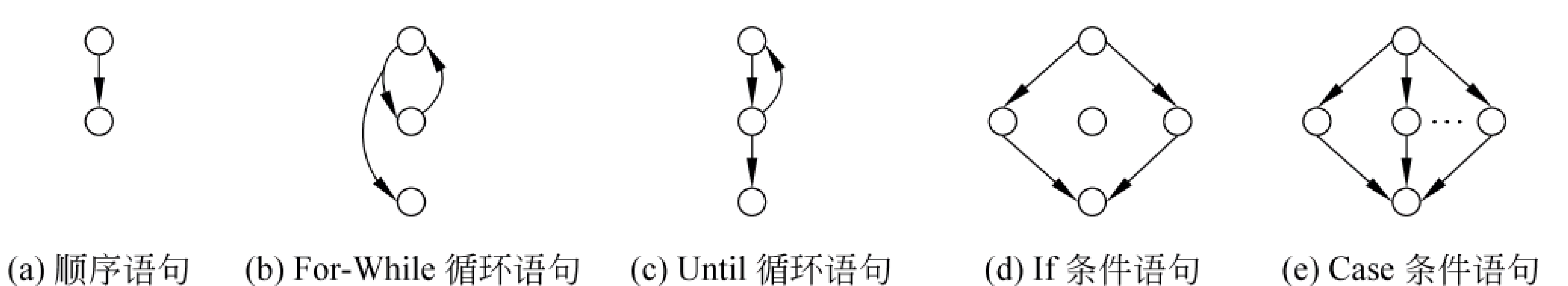


图 2-7 常见的控制流图

2. 程序环路复杂性

程序的环路复杂性是一种描述程序内部逻辑复杂度的标准，该标准运用基本路径方法，给出了程序独立路径集中的独立路径条数，这是确保程序中每个可执行语句至少执行一次所必需的测试用例数目的上界。

给定一个控制流图 G，设其环形复杂度为 $V(G)$，在这里介绍三种常见的计算方法来求解 $V(G)$。

(1) $V(G)=E-N+2$，其中 E 是控制流图 G 中边的数量，N 是控制流图中结点的数目；

(2) $V(G)=P+1$，其中 P 是控制流图 G 中判断结点(即包含条件的结点)的数目；

(3) $V(G)=A$，其中 A 是控制流图 G 中区域的数目。由边和结点围成的部分叫做区域，当在控制流图中计算区域的数目时，控制流图外的部分也应记为一个区域。

2. 独立路径测试

独立路径测试是在程序控制流图的基础上，通过分析控制流图的环路复杂性，导出可执行的独立路径集合，从而设计相应的测试用例。由对逻辑覆盖测试的讨论可知，对于一个较为复杂的程序要做到完全的路径覆盖测试是难以实现的。既然完整的路径覆

盖测试无法达到，那么可以对程序的所有独立路径进行测试，保证被测程序的每条可执行独立路径至少被执行一次。

从控制流图来看，一条独立路径是至少包含有一条在其他独立路径中从未有过的边的路径。路径可以用控制流图中的结点序列来表示。例如，如图 2-8 所示的控制流图中，一组独立的路径是：

Path1: 1→11

Path2: 1→2→3→4→5→10→1→11

Path3: 1→2→3→6→8→9→10→1→11

Path4: 1→2→3→6→7→9→10→1→11

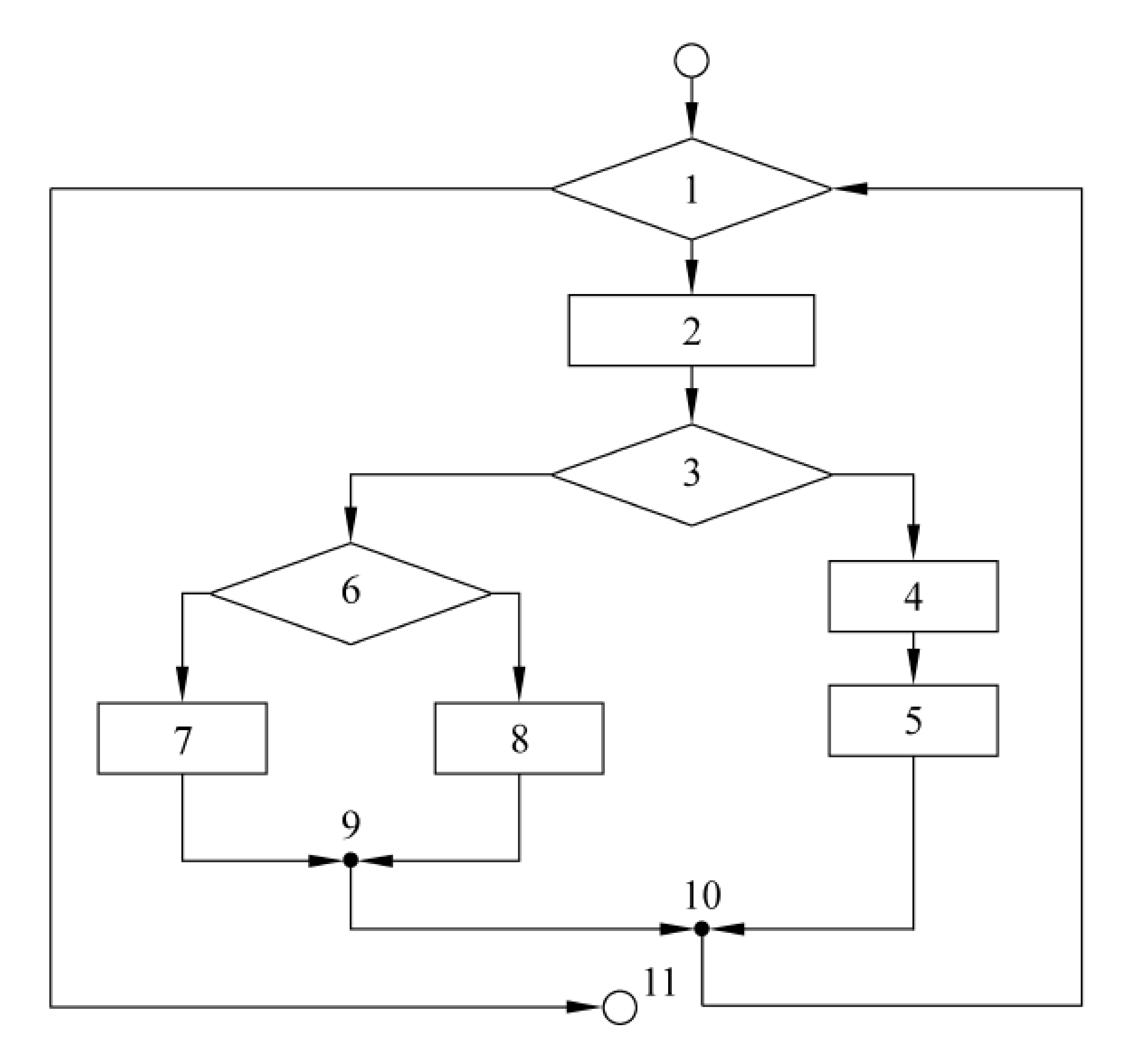


图 2-8 控制流图示例

路径 Path1、Path2、Path3、Path4 组成了控制流图的一个独立路径集。通常，独立路径集并不唯一确定。独立路径测试包括 3 个步骤：

① 导出程序控制流图；

② 求出程序环形复杂度；

③ 基于独立路径集设计测试用例。

下面通过一个 Java 语言程序实例来具体说明独立路径测试的设计流程。这段程序是统计一行字符中有多少个单词，单词之间用空格分隔开。

程序 2-2：

```
public static void main(String args[])
{
   int num1=0, num2=0, score=100;
   Scanner scan=new Scanner(System.in);
   int i=scan.nextInt();
   String str=scan.next();
   while (i<5)
```

```
8        {
9          if (str.equals("T"))
10           num1++;
11         else if (str.equals("F"))
12         {
13           score=score-10;
14           num2++;
15         }
16         i++;
17       }
18       System.out.println("num1="+num1+",num2="+num2+",score="+score);
19    }
```

1）导出程序控制流图

根据源代码可以导出程序的控制流图，如图 2-9 所示。每个圆圈代表控制流图的结点，可以表示一个或多个语句。圆圈中的数字对应程序中某一行的编码，箭头代表边的方向，即控制流方向。

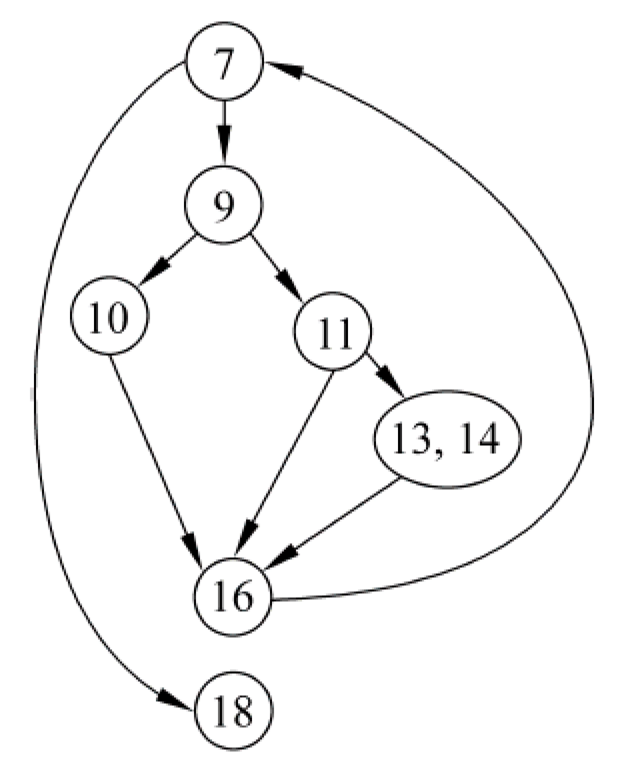


图 2-9 程序控制流图

2）求出程序环形复杂度

根据程序环形复杂度的计算公式，求出程序路径集合中的独立路径数目。

公式 1：$V(G)=9-7+2$。其中 10 是控制流图 G 中边的数量，8 是控制流图中结点的数目。

公式 2：$V(G)=3+1$。其中 3 是控制流图 G 中判断结点的数目（循环条件结点也属于判断结点）。

公式 3：$V(G)=4$。其中 4 是控制流图 G 中区域的数目。

因此，控制流图 G 的环形复杂度是 4，就是说至少需要 4 条独立路径组成独立路径集合，并由此得到能够覆盖所有程序语句的测试用例。

3）设计测试用例

根据上面环形复杂度的计算结果，源程序的独立路径集合中有 4 条独立路径：

```
Path1: 7→18
Path2: 7→9→10→16→7→18
Path3: 7→9→11→16→7→18
Path4: 7→9→11→13→14→16→7→18
```

根据上述 4 条独立路径，设计了测试用例组 9，如表 2-22 所示。测试用例组 9 中的 4 个测试用例作为程序输入数据，能够遍历这 4 条独立路径。对于源程序中的循环体，测试用例组 9 中的输入数据使其执行零次或一次。

注意：如果程序中的条件判断表达式是由一个或多个逻辑运算符(or、and、not)连接的复合表达式，则需要变换为一系列只有单个条件的嵌套判断。如程序 2-3 的伪代码所示：

表 2-22　测试用例组 9

测试用例	输　入		预 期 输 出			执行路径
	i	str	num1	num2	score	
Test Case 1	5	"T"	0	0	100	路径 1
Test Case 2	4	"T"	1	0	100	路径 2
Test Case 3	4	"A"	0	0	100	路径 3
Test Case 4	4	"F"	0	1	90	路径 4

程序 2-3：

```
1    if (a or b)
2    then
3      procedure x
4    else
5      procedure y
6    …
```

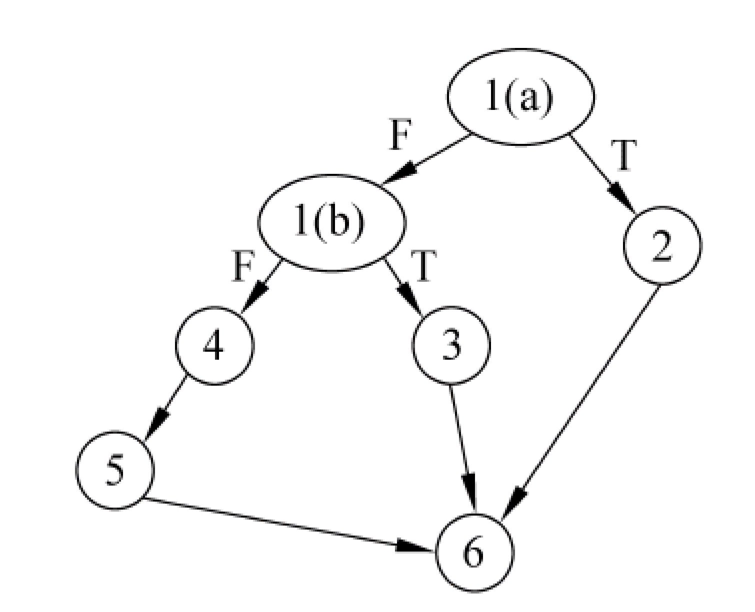


图 2-10　程序控制流图

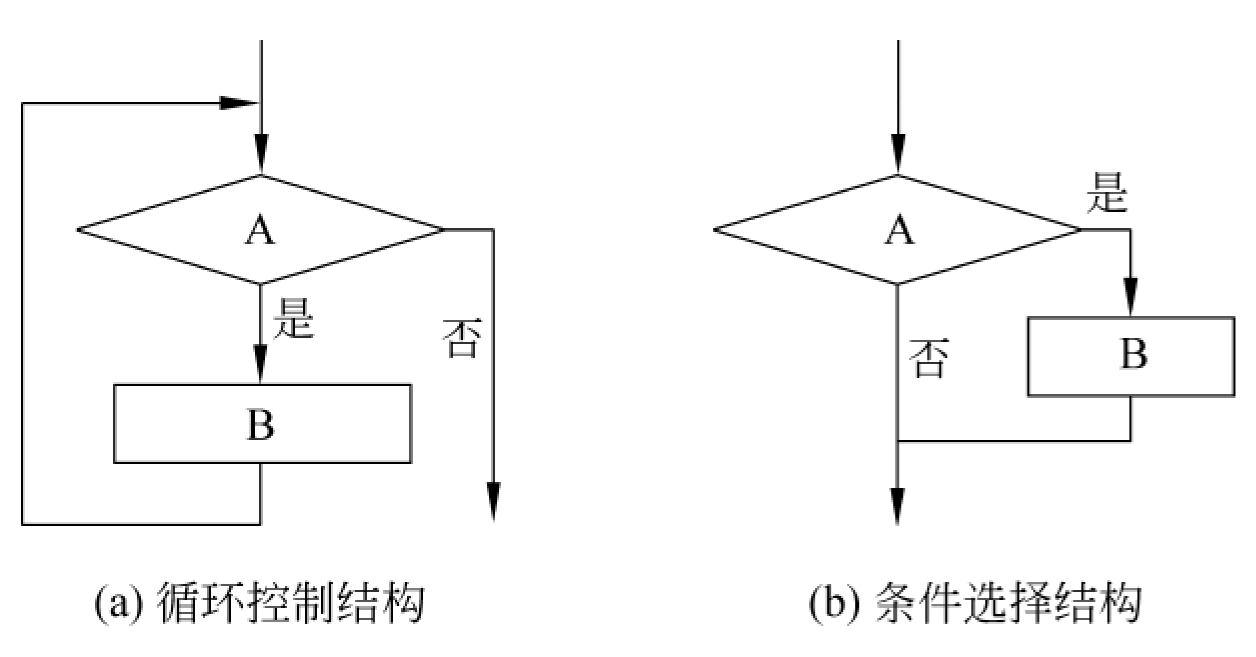
对应的控制流图如图 2-10 所示，程序行 1 的 a、b 都是独立的判断结点，还有程序行 4 也是判断结点，所以共计3 个判断结点。图 2-10 的环形复杂度为 $V(G)=3+1$，其中 3 是图中判断结点的数目。

3. Z 路径覆盖测试

和独立路径选择一样，Z 路径覆盖也是一种常见的路径覆盖方法。为解决实际操作中路径覆盖难以实现问题，必须舍弃一些次要因素，简化循环结构，从而极大地减少路径的数量，使得覆盖这些有限的路径成为可能。采用简化循环方法的路径覆盖就是 Z 路径覆盖。

所谓简化循环就是减少循环的次数，不考虑循环体的形式和复杂度如何，也不考虑循环体实际上需要执行多少次，只考虑通过循环体零次和一次这两种情况。这里的零次循环是指跳过循环体，从循环体的入口直接到循环体的出口。通过一次循环体是指检查循环初始值。根据简化循环的思路，循环要么执行，要么跳过，这和判定分支的效果是一样的。可见，简化循环就是将循环结构变成选择结构。

如图 2-11 所示的例子，图(a)表示了一种典型的循环控制结构，A 为循环条件，B 为循环体，若限定循环只执行零次或一次，则该控制结构就退化成图(b)所示的条件选择结构。



(a) 循环控制结构　　(b) 条件选择结构

图 2-11　Z 路径覆盖示例

第 3 章



软件测试过程

本章介绍了软件测试过程。软件测试也存在着生命周期的概念，即软件测试生命周期，它包括测试计划，测试设计，实施测试以及测试评估等几个部分。

3.1 概　　述

在统一软件开发过程(Rational Unified Process，RUP)中，测试生命周期分为测试计划、测试设计、测试开发、测试执行、缺陷跟踪和评估测试等，如图 3-1 所示。

软件测试过程中必需的基本测试活动及其产生的结果：

1. 拟定软件测试计划(plans)

定义测试项目的阶段，便于对项目进行适当的评估与控制，测试计划包括测试需求，测试策略、测试资源和测试进度。

2. 编制软件测试大纲(outlines)

测试大纲主要说明要测试的内容、参考资料、测试的阶段(内部测试、Alpha 测试、Beta 测试)、要进行哪些类型的测试(功能测试、性能测试、界面测试)、进度安排、人员和资源的需求等。

3. 设计和生成测试用例(test case generation)

设计测试用例及测试过程阶段，是验证测试需求被测试到的最有效方法。

制定测试计划
设计测试
实施测试
执行测试
评估测试
回归测试

图 3-1　软件测试过程

4. 实施测试(execution)

实施测试包含测试的执行过程(如单元测试、集成测试、系统测试等)，是对测试设计

阶段已被定义的测试进行创建或修正的阶段。

5. 生成软件测试报告(software testing reports)

对被测软件进行一系列测试并记录结果,分析测试结果并判断测试的标准是否被满足,一般生成软件问题报告(software problem report)和测试结果报告(test result reports)[6]。

3.2 软件测试计划

测试计划就是描述所有要完成的测试工作,包括被测试项目的背景、目标、范围、方式、资源、进度安排、测试组织,以及与测试有关的风险等方面。测试计划规定测试活动的范围、方法、资源和进度;明确正在测试的项目、要测试的特性,要执行的测试任务、每个任务的负责人,以及与计划相关的风险。测试计划采用的形式是书面文档。

3.2.1 制定测试计划的作用和原则

1. 制定软件测试计划的作用

(1) 使软件测试工作进行更顺利。
(2) 促进项目参加人员彼此的沟通。
(3) 及早发现和修正软件规格说明书的问题。
(4) 使软件测试工作更易于管理。

2. 测试计划制定原则

制定测试计划是软件测试中最有挑战性的一个工作。以下原则将有助于制定测试计划工作。
(1) 制定测试计划应尽早开始。
(2) 保持测试计划的灵活性。
(3) 保持测试计划简洁和易读。
(4) 尽量争取多渠道评审测试计划。
(5) 计算测试计划的投入。

3. 一份好的测试计划书应具备的特点

(1) 能有效地引导整个软件测试工作正常运行,并配合编程部门,保证软件质量,按时将产品推出。
(2) 能使测试高效地进行,即能在较短的时间内找出尽可能多的软件缺陷。
(3) 提供了明确的测试目标、测试策略、具体步骤及测试标准。
(4) 既强调测试重点,也重视测试的基本覆盖率。
(5) 尽可能充分利用公司现有的、可以提供给测试部门的人力和物力资源,而且是可

行的。

(6) 列举的所有数据都必须是准确的,如外部软硬件的兼容性所要求的数据、输入输出数据等。

(7) 对测试工作的安排有一定的灵活性,可以应付一些突然的变化情况,如当时间安排或产品出现的一些变化的时候。

3.2.2 测试计划的内容

制定测试计划时,由于各软件公司的背景不同,测试计划文档也略有差异。实践表明,制定测试计划时,使用正规化文档通常比较好。为了使用方便,在这里给出 IEEE 软件测试计划文档模板,如图 3-2 所示。

IEEE829-1998 软件测试文档编制标准
软件测试计划文档模板

目录
1. 测试计划标识符
2. 介绍
3. 测试项
4. 需要测试的功能
5. 方法(策略)
6. 不需要测试的功能
7. 测试项通过/失败的标准
8. 测试中断和恢复的规定
9. 测试完成所提交的材料
10. 测试任务
11. 环境需求
12. 测试人员的工作职责
13. 人员安排与培训需求
14. 进度表
15. 潜在的问题和风险
16. 审批

图 3-2 IEEE 软件测试计划文档模板

根据 IEEE829-1998 软件测试文档编制标准的建议,测试计划包含了 16 个大纲要项,简要说明如下。

1. 测试计划标识符

一个测试计划标识符是一个由公司生成的唯一值,它用于标识测试计划的版本、等级,以及与该测试计划相关的软件版本。

2. 介绍

在测试计划的介绍部分主要是测试软件基本情况的介绍和测试范围的概括性描述。

3. 测试项目

测试项部分主要是纲领性描述在测试范围内对哪些具体内容进行测试，确定一个包含所有测试项在内的一览表。有功能测试、设计测试、整体测试。

IEEE 标准中指出，可以参考需求规格说明、用户指南、操作指南、安装指南、与测试项相关的报告来完成测试项。

4. 需要测试的功能

列出待测对象的单项功能及组合功能。

5. 不需要测试的功能

列出不测试的单项功能及组合功能并说明不予测试的理由。

6. 测试方法(策略)

测试策略描述测试小组用于测试整体和每个阶段的方法。要描述如何公正、客观地开展测试，要考虑模块、功能、整体、系统、版本、压力、性能、配置和安装等各个因素的影响，要尽可能地考虑到细节，越详细越好，并制作测试记录文档的模板，为即将开始的测试做准备，主要完成以下事项。

(1) 确定要使用的测试技术和工具。
(2) 确定测试完成的标准。
(3) 对于影响资源分配的特殊考虑，如测试与外部接口等。

7. 测试项通过/失败的标准

测试计划中这一部分给出了“测试项”中描述的每一个测试项通过/失败的标准。正如每个测试用例都需要一个预期的结果一样，每个测试项同样都需要一个预期的结果。

下面是通过/失败的标准的一些例子：

- 通过测试用例所占的百分比；
- 缺陷的数量、严重程度和分布情况；
- 测试用例覆盖；
- 用户测试的成功结论；
- 文档的完整性；
- 性能标准。

8. 测试中断和恢复的规定

测试计划中这一部分给出了测试中断和恢复的标准。常用的测试中断标准如下：

- 关键路径上的未完成任务；
- 大量的缺陷；
- 严重的缺陷；

- 不完整的测试环境;
- 资源短缺。

9. 测试完成所提交的材料

测试完成所提交的材料包含了测试工作开发设计的所有文档、工具等。例如,测试计划、测试设计规格说明、测试用例、测试日志、测试数据、自定义工具、测试缺陷报告和测试总结报告等。

10. 测试任务

测试计划中这一部分给出了测试工作所需完成的一系列任务。在这里还列举了所有任务之间的依赖关系和可能需要的特殊技能。

11. 测试所需的资源

测试所需的资源是实现测试策略所必需的。例如:

- 人员——人数、经验和专长。他们是全职、兼职、业余的工作人员还是学生?
- 设备——计算机、测试硬件、打印机、测试工具等。
- 办公室和实验室空间——在哪里?空间有多大?怎样排列?
- 软件——字处理程序、数据库程序和自定义工具等。
- 其他资源——软盘、电话、参考书、培训资料等。

12. 测试人员的工作职责

测试人员的工作职责是明确指出了测试任务和测试人员的工作责任。

有时测试需要定义的任务类型不容易分清,不像程序员所编写的程序那样明确。复杂的任务可能有多个执行者,或者由多人共同负责。

13. 人员安排与培训需求

前面讨论的测试人员的工作职责是指哪类人员(管理、测试和程序员等)负责哪些任务。人员安排与培训需求是指明确测试人员具体负责软件测试的哪些部分、哪些可测试性能,以及需掌握的技能等。实际责任表会更加详细,确保软件的每一部分都有人进行测试。每一个测试员都会清楚地知道自己应该负责什么,而且有足够的信息开始设计测试用例。

培训需求通常包括学习如何使用某个工具、测试方法、缺陷跟踪系统、配置管理,或者与被测试系统相关的业务基础知识。培训需求各个测试项目会各不相同,它取决于具体项目的情况。

14. 测试进度表

测试进度是围绕着包含在项目计划中的主要事件(如文档、模块的交付日期,接口的可用性等)来构造的。

作为测试计划的一部分,完成测试进度计划安排,可以为项目管理员提供信息,以便更好地安排整个项目的进度。表3-1给出了一个例子。

表3-1 相对日期的测试进度

测试任务	开始日期	期限	测试任务	开始日期	期限
测试计划完成	说明书完成之后7天	4周	第2阶段测试通过	Beta构造	6周
测试案例完成	测试计划完成	12周	第3阶段测试通过	发布构造	4周
第1阶段测试通过	代码计划完成	6周			

进度安排会使测试过程容易管理。通常,项目管理员或者测试管理员最终负责进度安排,而测试人员参与安排自己的具体任务。

15. 风险及应急措施

软件测试人员要明确地指出计划过程中的风险,并与测试管理员和项目管理员交换意见。这些风险应该在测试计划中明确指出,在进度中予以考虑。有些风险是真正存在的,而有些最终证实是无所谓的,重要的是尽早明确指出,以免在项目晚期发现时感到惊慌。

一般而言,大多数测试小组都会发现自己的资源有限,不可能穷尽测试软件所有方面。如果能勾画出风险的轮廓,将有助于测试人员排定待测试项的优先顺序,并且有助于集中精力去关注那些极有可能发生失效的领域。下面列出一些潜在的问题和风险:

- 不现实的交付日期;
- 与其他系统的接口;
- 处理巨额现金的特征;
- 极其复杂的软件;
- 有过缺陷历史的模块;
- 发生过许多或者复杂变更的模块;
- 安全性、性能和可靠性问题;
- 难于变更或测试的特征。

16. 审批

审批人应该是有权宣布已经为转入下一个阶段做好准备的某个人或某几个人。测试计划审批部分一个重要的部件是签名页。审批人除了在适当的位置签署自己的名字和日期外,还应该签署表明他们是否建议通过评审的意见。

3.3 测试用例

Grenford J. Myers在*The Art of Software Testing*一书中提出:一个好的测试用例是指很可能找到迄今为止尚未发现的错误的测试用例,由此可见测试用例设计工作在整个测试过程中的地位。

影响软件测试的因素很多，例如软件本身的复杂程度、开发人员（包括分析、设计、编程和测试的人员）的素质、测试方法和技术的运用等。因为有些因素是客观存在的，无法避免。有些因素则是波动的、不稳定的，例如开发队伍是流动的，有经验的走了，新人不断补充进来；某位工作人员的情绪受到影响等。如何保障软件测试质量的稳定？有了测试用例，无论是谁来测试，参照测试用例实施，都能保障测试的质量。可以把人为因素的影响减少到最小。即便最初的测试用例考虑不周全，随着测试的进行和软件版本更新，也将日趋完善。

因此，测试用例的设计和编制是软件测试活动中最重要的。测试用例是测试工作的指导，是软件测试的必须遵守的准则，更是软件测试质量稳定的根本保障。

3.3.1　测试用例定义

测试用例（test case）目前没有经典的定义。比较通常的说法是：指对一项特定的软件产品进行测试任务的描述，体现测试方案、方法、技术和策略，内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。

IEEE 610.12 给出测试用例的定义如下：

（1）测试用例是一组输入（运行前提条件）和为某特定的目标而生成的预期结果及与之相关的测试规程的一个特定集合。

（2）测试用例是一个详细说明测试的输入、期望输出和为一测试项所准备的一组执行条件。

其中，定义（1）给出了测试用例的实质，定义（2）是测试用例的存在方式。

3.3.2　测试用例在软件测试中的作用

1. 指导测试的实施

测试用例主要适用于集成测试、系统测试和回归测试。在实施测试时测试用例作为测试的标准，测试人员一定要按照测试用例严格按用例项目和测试步骤逐一实施测试，并将测试情况记录在测试用例管理软件中，以便自动生成测试结果文档。

根据测试用例的测试等级，集成测试应测试哪些用例，系统测试和回归测试又该测试哪些用例，在设计测试用例时都已作明确规定，实施测试时测试人员不能随意作变动。

2. 规划测试数据的准备

测试实践中测试数据是与测试用例相分离的。按照测试用例配套准备一组或若干组测试原始数据，以及标准测试结果。除正常数据之外，还必须根据测试用例设计大量边缘数据和错误数据。

3. 保证软件的可维护性和可复用性

软件功能模块的通用化和复用化使软件易于开发，只需要修改少部分的测试用例便可以开展测试，测试用例的反复使用提高了测试的效率，缩短了项目周期。

4. 评估测试结果的度量基准

完成测试实施后需要对测试结果进行评估，并且编制测试报告。判断软件测试是否完成、衡量测试质量的优劣需要一些量化的结果。例如，测试覆盖率是多少、测试合格率是多少、重要测试合格率是多少等。以前统计基准是软件模块或功能点，显得过于粗糙。采用测试用例作度量基准更加准确、有效。

5. 分析缺陷的标准

通过收集缺陷，对比测试用例和缺陷数据库，从而分析缺陷是漏测还是缺陷复现。漏测反映了测试用例的不完善，应立即补充相应测试用例，最终达到逐步完善软件质量。而已有相应测试用例，则反映实施测试或变更处理存在问题。

3.3.3 测试用例设计的基本原则

测试用例在设计时应遵循以下原则：

1. 测试用例的代表性

能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的以及极限的输入数据、操作和环境设置等。

2. 测试结果的可判定性

测试执行结果的正确性是可判定的，每一个测试用例都应有相应的期望结果。

3. 测试结果的可再现性

对同样的测试用例，系统的执行结果应当是相同的。

3.3.4 测试用例设计应注意的问题

软件测试用例是为了有效发现软件缺陷而编写的包含测试目的、测试步骤、期望测试结果的特定集合。正确认识和设计软件测试用例可以提高软件测试的有效性，便于测试质量的度量，增强测试过程的可管理性。在实际软件项目测试过程中，由于对软件测试用例的作用和设计方法的理解不同，测试人员(特别是刚从事软件测试的新人)对软件测试用例存在不少错误的认识，给实际软件测试带来了负面影响。以下几点是经常遇到的错误做法。

1. 把测试输入数据设计方法等同于测试用例设计方法

现在不少人认为测试用例设计就是如何确定测试的输入数据，从而掩盖了测试用例设计内容的丰富性和技术的复杂性。对于软件功能测试和性能测试，确定测试的输入数据很重要，它决定了测试的有效性和测试的效率。但是，测试用例中输入数据的确定方

法只是测试用例设计方法的一个子集，除了确定测试输入数据之外，测试用例的设计还包括如何根据测试需求、设计规格说明等文档确定测试用例的设计策略、设计用例的表示方法和组织管理形式等问题。

在设计测试用例时，需要综合考虑被测软件的功能、特性、组成元素、开发阶段（里程碑）、测试用例组织方法（是否采用测试用例的数据库管理）等内容。具体到设计每个测试用例而言，可以根据被测模块的最小目标，确定测试用例的测试目标；根据用户使用环境确定测试环境；根据被测软件的复杂程度和测试用例执行人员的技能确定测试用例的步骤；根据软件需求文档和设计规格说明确定测试用例期望的执行结果。

2. 强调测试用例设计得越详细越好

在确定测试用例设计目标时，一些项目管理人员强调测试用例"越详细越好"。具体表现在两个方面：一是尽可能设计足够多的设计用例，测试用例的数量越多越好；二是测试用例尽可能包括测试执行的详细步骤，达到"任何一个人都可以根据测试用例执行测试"，追求测试用例越详细越好。

这种做法和观点最大的危害就是耗费了很多的测试用例设计时间和资源，可能等到测试用例设计、评审完成后，留给实际执行测试的时间就所剩无几了。因为当前软件公司的项目团队在规划测试阶段，分配给测试的时间和人力资源是有限的，而软件项目的成功要坚持"质量、时间、成本"的最佳平衡，没有足够多的测试执行时间，就无法发现更多的软件缺陷，测试质量更无从谈起了。

编写测试用例的根本目的是有效地找出软件可能存在的缺陷，为了达到这个目的，需要分析被测试软件的特征，运用有效的测试用例设计方法，尽量使用较少的测试用例，同时满足合理的测试需求覆盖，从而达到"少花时间多办事"的效果。

3. 追求测试用例设计"一步到位"

一些人认为设计测试用例是一次性投入，测试用例设计一次就可以了，片面追求测试设计的"一步到位"。这种认识造成的危害性使设计出的测试用例缺乏实用性。

任何软件项目的开发过程都处于不断变化过程中，用户可能对软件的功能提出新需求，设计规格说明相应地更新，软件代码不断细化。设计软件测试用例与软件开发设计并行进行，必须根据软件设计的变化，对软件测试用例进行内容的调整，数量的增减，增加一些针对软件新增功能的测试用例，删除一些不再适用的测试用例，修改那些模块代码更新了的测试用例。

软件测试用例设计只是测试用例管理的一个过程，除此之外，还要对其进行评审、更新、维护，以便提高测试用例的"新鲜度"，保证"可用性"。因此，软件测试用例也要坚持与时俱进的原则。

4. 让测试新人设计测试用例

软件测试用例设计是软件测试的中高级技能，不是每个人（尤其是测试新人）都可以编写的，测试用例编写者不仅要掌握软件测试的技术和流程，而且要对被测软件的设计、

功能规格说明、用户试用场景以及程序/模块的结构都有比较透彻的理解。让测试新人设计测试用例是一种高风险的测试组织方式,它带来的不利后果是设计出的测试用例对软件功能和特性的测试覆盖性不高,编写效率低,审查和修改时间长,可重用性差。

因此,实际测试过程中,通常安排经验丰富的测试人员进行测试用例设计。测试新人可以从执行测试用例开始。随着项目的不断开展,测试人员对测试技术和被测软件的不断熟悉,可以积累测试用例的设计经验,编写测试用例。

3.3.5 测试用例的编写标准

在编写测试用例过程中,需要参考和规范一些基本的测试用例编写标准,在 ANSI/IEEE 829 标准中列出了和测试计划相关的测试用例编写规范和模板。标准模板中主要元素如下。

1. 标识符(identification)

每个测试用例应该有一个唯一的标识符,它将成为所有和测试用例相关的文档及表格引用和参考的基本元素,这些文档及表格包括设计规格说明书、测试日志表、测试报告等。

2. 测试项(test item)

测试用例应该准确地描述所需要测试的项及其特征,测试项应该比测试设计说明中所列出的特性描述更加具体,例如做 Windows 计算器应用程序的窗口测试,测试对象是整个的应用程序用户界面,其测试项就应该是应用程序的界面和特性要求,如窗口缩放测试、界面布局、菜单等。

3. 输入标准(input criteria)

用来执行测试用例的输入要求。这些输入可能包括数据、文件或者操作(如,鼠标单击,键盘按键处理等),必要的时候,相关的数据库、文件也应被罗列。

4. 输出标准(output criteria)

用于标识按照指定的环境和输入标准得到的期望输出结果。尽可能提供适当的系统规格说明来证明期望的结果。

5. 测试用例之间的关联

用于标识该测试用例与其他的测试或测试用例之间的依赖关系。在测试的实际过程中,很多的测试用例并不是单独存在的,它们之间可能有某种依赖关系,例如,用例 A 需要在 B 测试结果正确的基础上才能进行,此时需要在 A 的测试用例中表明对 B 的依赖性,从而保证测试用例的严谨性。

表 3-2 所示是 ANSI/IEEE 829 标准给出的测试用例编写的表格形式,在编写测试用例时可以用来参考。

表 3-2 测试用例

编号：

编制人：	审定人：	时间：
软件名称：	编号/版本：	
测试用例：		
用例编号：		
参考信息(参考的文档及章节号或功能项)：		
输入说明(列出选用的输入项)：		
输出说明(逐条与输入项对应,列出预期输出)：		
环境要求(测试要求的软件、硬件、网络要求)：		
特殊规程要求：		
操作步骤：		
用例间的依赖关系：		
用例产生的测试程序限制：		

3.4 软件测试的过程模型

目前主流的软件开发模型主要有：瀑布模型、原型模型、螺旋模型、增量模型、渐进模型、快速软件开发(RAD)以及 Rational 统一过程(RUP)等。这些模型对于软件开发过程具有很好的指导作用,而软件测试是在软件投入运行前,对软件需求分析、设计规格说明和编码实现的最终审定,在软件生存期中占据着极其重要的位置。很显然,表现在程序中的故障,并不一定是由编码所引起的,很可能是详细设计、概要设计阶段,甚至是需求分析阶段的问题引起的,即使针对源程序进行测试,所发现故障的根源也可能存在于开发前期的各个阶段。解决问题、排除故障也必须追溯到前期的工作。非常遗憾的是,在这些过程方法中,并没有充分强调测试的价值,也没有给测试以足够的重视,利用这些模型无法更好地指导测试实践。软件测试是与软件开发紧密相关的一系列有计划的系统性的活动,显然软件测试也需要测试模型去指导实践。下面对主要的模型做一些简单的介绍。

3.4.1 V模型

1. V 模型建立

V 模型是最具有代表意义的测试模型。在传统的开发模型中,比如瀑布模型,人们通常把测试过程作为在需求分析、概要设计、详细设计和编码全部完成后的一个阶段。尽管有时测试工作会占用整个项目周期的一半时间,但是有人仍然认为测试只是一个收尾工作,而不是主要过程。V 模型的推出就是对此种认识的改进。V 模型是软件开发瀑布模型的变种,如图 3-3 所示。它反映了测试活动与分析和分析与设计的关系,从左到右,描述了基本的开发过程和测试行为,非常明确地标明了测试过程中存在的不同级别,

并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系,图中的箭头代表了时间方向,左边下降的是开发过程各阶段,与此相对应的是右边上升的部分,即各测试过程的各个阶段。

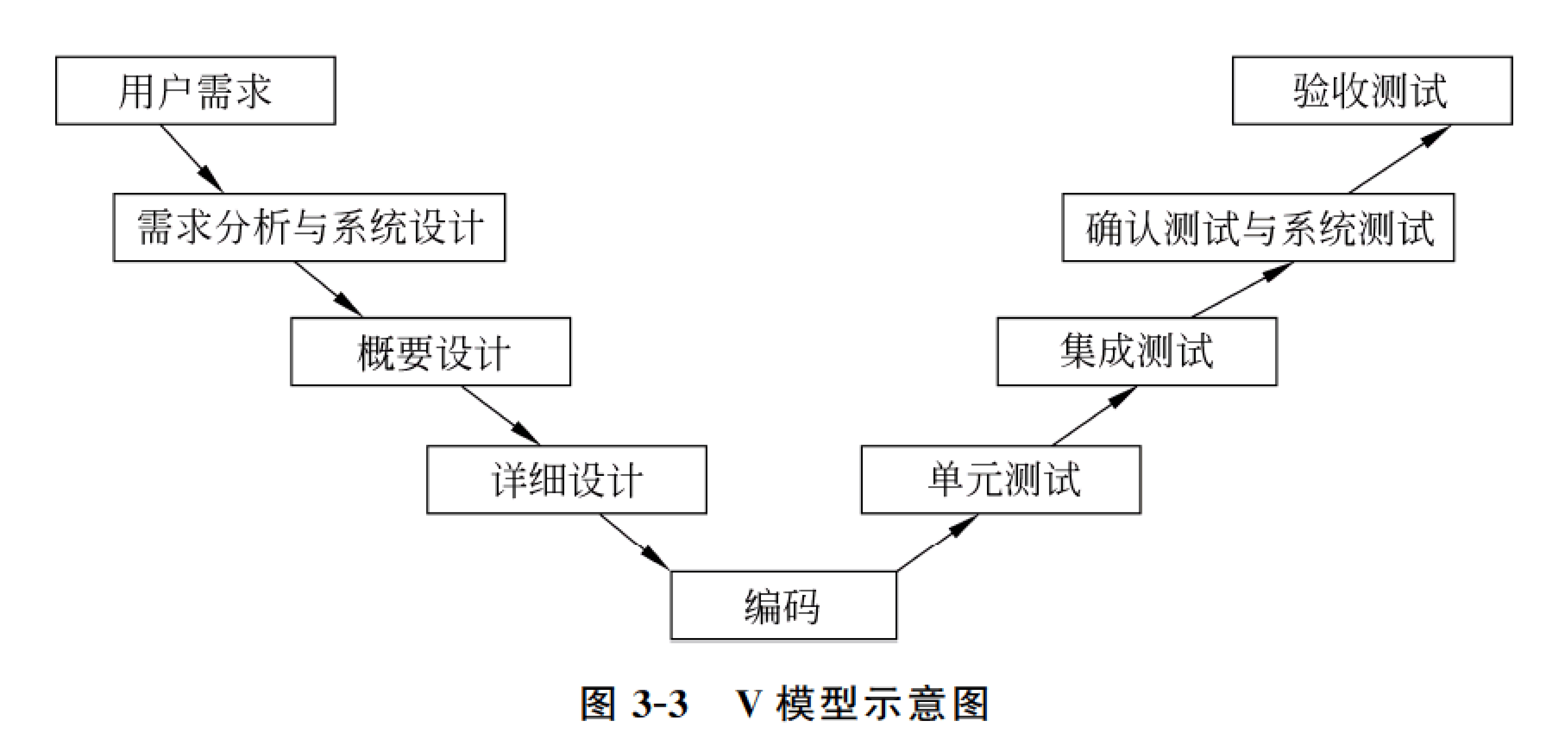


图 3-3 V 模型示意图

2. V 模型应用

V 模型的软件测试策略既包括低层测试又包括高层测试,低层测试是为了源代码的正确性;高层测试是为了使整个系统满足用户的需求。

V 模型指出,单元和集成测试是验证程序设计,开发人员和测试组应检测程序的执行是否满足软件设计的要求。首先对每一个程序模块进行单元测试,以确保每个模块能正常工作。单元测试大多采用白盒测试方法,尽可能发现并消除模块内部在逻辑和功能上的故障及缺陷,然后,把已测试过的模块组装起来,形成一个完整的软件后进行集成测试,以检测和排除与软件设计相关的程序结构问题;集成测试大多采用黑盒测试方法来设计测试用例。确认测试以规格说明规定的需求为尺度,检验开发的软件能否满足所有的功能和性能要求。确认测试完成以后,给出的应该是合格的软件产品,但为了检验开发的软件是否能与系统的其他部分(如硬件、数据库及操作人员)协调工作,还需进行系统测试。系统测试应当验证系统设计,检测系统功能、性能的质量特性是否达到系统设计的指标。最后由测试人员和用户进行软件的验收测试,追溯软件需求说明书进行测试,以确定软件的实现是否满足用户需求或合同要求。

V 模型存在一定的局限性,它仅仅把测试过程作为在需求分析、概要设计、详细设计及编码之后的一个阶段。容易使人理解为测试是软件开发的最后的一个阶段,主要是针对程序进行测试寻找错误,而需求分析阶段隐藏的问题一直到后期的验收测试才被发现。

3.4.2 W 模型

1. W 模型建立

V 模型的局限性在于没有明确地说明早期的测试,不能体现"尽早地和不断地进行软件测试"的原则。在 V 模型中增加软件各开发阶段应同步进行的测试,被演化为一种

W 模型，因为实际上开发是“V”，测试也是与此相并行的“V”。基于“尽早地和不断地进行软件测试”的原则，在软件的需求和设计阶段的测试活动应遵循 IEEE std1012-1998《软件验证和确认(V&V)》的原则。

一个基于 V&V 原理的 W 模型示意如图 3-4 所示。

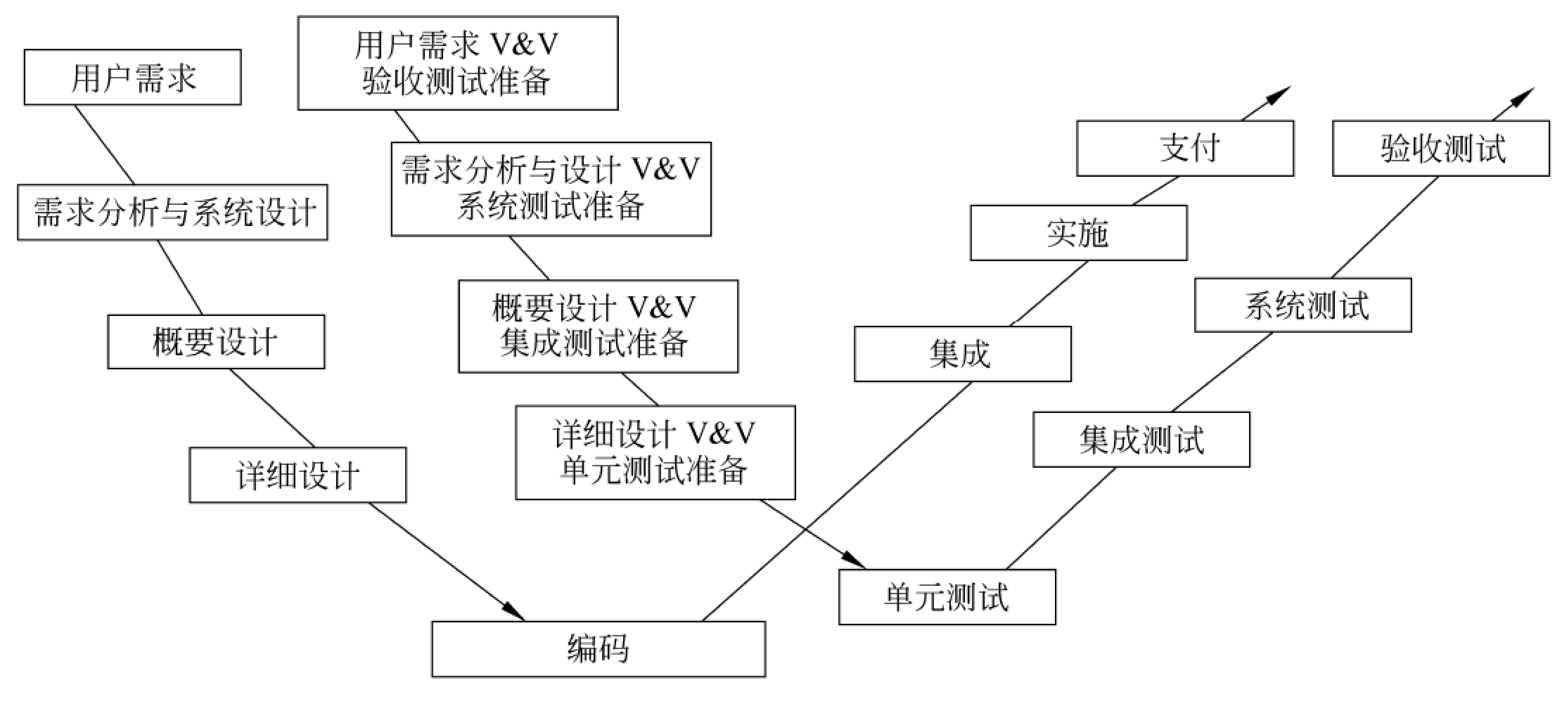


图 3-4 软件测试 W 模型

2. W 模型应用

相对于 V 模型，W 模型更科学。W 模型可以说是 V 模型的发展。它强调：测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、功能和设计同样要测试。只要相应的开发活动完成，就可以开始执行测试，可以说，测试与开发是同步进行的，从而有利于尽早地发现问题。以需求为例，需求分析一完成，就可以对需求进行测试，而不是等到最后才进行针对需求的验收测试。

如果测试文档能尽早提交，那么就有了更多的检查和检阅的时间，这些文档还可用于评估开发文档。另外还有一个很大的益处是，测试者可以在项目中尽可能早地面对规格说明书中的挑战。这意味着测试不仅仅是评定软件的质量，测试还可以尽可能早地找出缺陷所在，从而帮助改进项目内部的质量。参与前期工作的测试者可以预先估计问题和难度，这将可以显著地减少总体测试时间，加快项目进度。

根据 W 模型的要求，一旦有文档提供，就要及时确定测试条件，以及编写测试用例。这些工作对测试的各级别都有意义。当需求被提交后，就需要确定高级别的测试用例来测试这些需求。当概要设计编写完成后，就需要确定测试条件来查找该阶段的设计缺陷。

W 模型也有局限性。W 模型和 V 模型都把软件的开发视为需求、设计、编码等一系列串行活动。软件开发和测试保持一种线性的前后关系，需要有严格的指令表示上一阶段完全结束，才可以正式开始下一个阶段。这样就无法支持迭代、自发性以及变更调整。对于当前很多文档需要事后补充，或者根本没有文档的情况下(这已成为一种开发的文化)，开发人员和测试人员都面临同样的困惑。

3.4.3 H模型

1. H模型建立

V模型和W模型均存在一些不妥之处。首先，它们都把软件的开发视为需求、设计、编码等一系列串行的活动，而事实上，虽然这些活动之间存在相互牵制的关系，但在大部分时间内，它们是可以交叉进行的。虽然软件开发期望有清晰的需求、设计和编码阶段，但实践表明，严格的阶段划分只是一种理想状况。试问，有几个软件项目是在有了明确的需求之后才开始设计的呢？所以，相应的测试之间也不存在严格的次序关系。同时，各层次之间的测试也存在反复触发、迭代和增量关系。其次，V模型和W模型都没有很好地体现测试流程的完整性。

为了解决以上问题，提出了H模型。它将测试活动完全独立出来，形成一个完全独立的流程，将测试准备活动和测试执行活动清晰地体现出来。

2. H模型应用

H模型的简单示意如图3-5所示。

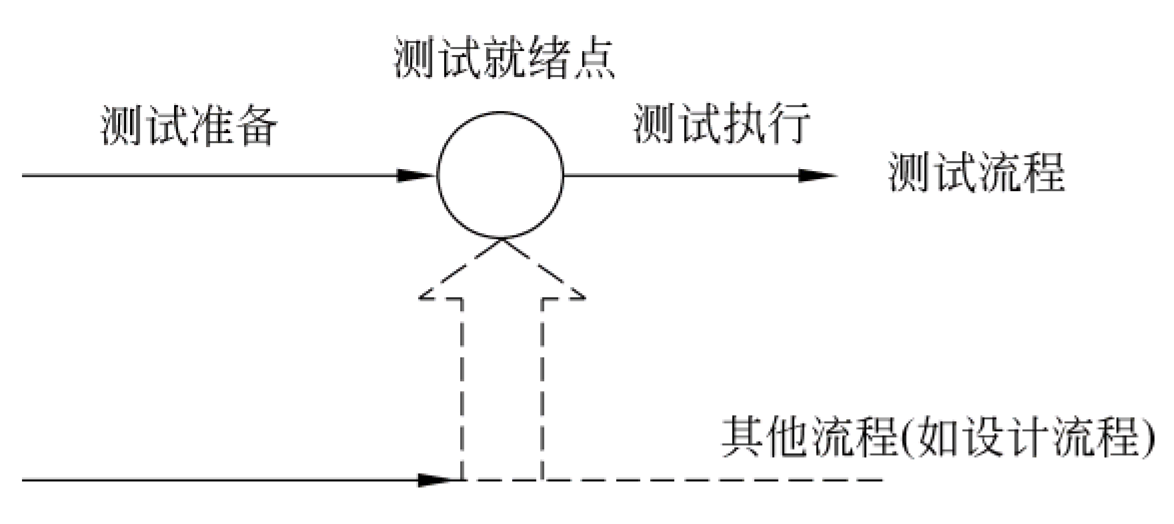


图3-5 软件测试H模型

这个示意图仅仅演示了在整个生产周期中某个层次上的一次测试“微循环”。图中的其他流程可以是任意开发流程，例如设计流程和编码流程，也可以是其他非开发流程，例如SQA流程，甚至是测试流程自身。也就是说，只要测试条件成熟了，测试准备活动完成了，测试执行活动就可以（或者说需要）进行了。

概括地说，H模型揭示了：

- 软件测试不仅仅指测试的执行，还包括很多其他的活动。
- 软件测试是一个独立的过程，贯穿产品整个生命周期，与其他流程并发地进行。
- 软件测试要尽早准备，尽早执行。
- 软件测试是根据被测物的不同而分层次进行的。不同层次的测试活动可以是按照某个次序先后进行的，但也可能是反复的。

在H模型中，软件测试模型是一个独立的流程，贯穿于整个产品周期，与其他流程并发地进行。当某个测试时间点就绪时，软件测试即从测试准备阶段进入测试执行阶段。

3.4.4 各种测试模型的使用

前面介绍了几种典型的测试模型,应该说这些模型对指导测试工作的进行具有重要的意义,但任何模型都不是完美的。我们应该尽可能地去应用模型中对项目有实用价值的方面,但不强行地为使用模型而使用模型,否则没有实际意义。

在这些模型中,V 模型强调了在整个软件项目开发中需要经历的若干个测试级别,而且每一个级别都与一个开发级别相对应,但它忽略了测试的对象不应该仅仅包括程序,或者说它没有明确地指出应该对软件的需求、设计进行测试,而这一点在 W 模型中得到了补充。W 模型强调了测试计划等工作的先行以及对系统需求和系统设计的测试,但 W 模型和 V 模型一样也没有专门对软件测试流程予以说明。而随着软件质量要求越来越为大家所重视,软件测试也逐步发展成为一个独立于软件开发部的组织,就每一个软件测试的细节而言,它都有一个独立的操作流程。比如,现在的第三方测试,就包含了从测试计划和测试案例编写,到测试实施以及测试报告编写的全过程。这个过程在 H 模型中得到了相应的体现,表现为测试是独立的。也就是说,只要测试前提具备了,就可以开始进行测试了。

因此,在实际的工作中,我们要灵活地运用各种模型的优点,在 W 模型的框架下,运用 H 模型的思想进行独立地测试,并同时将测试与开发紧密结合,寻找恰当的就绪点开始测试并反复迭代测试,最终保证按期完成预定目标。

3.5 软件测试实施过程

软件测试的实施过程一般经历如下三个阶段:

(1) 初测期。测试主要功能和关键的执行路径,排除主要障碍。

(2) 细测期。依据测试计划和测试大纲、测试用例,逐一测试各个功能、每个特性、性能、用户界面、兼容性、可用性等;预期可发现大量不同性质、不同严重程度的错误和问题。

(3) 回归测试期。系统已达到稳定,在一轮测试中发现的错误已十分有限;复查已知错误的纠正情况,确认未引发任何新的错误时,终结回归测试。

软件测试实施过程可分为单元测试、集成测试、确认测试、系统测试、验收测试和回归测试。

3.5.1 单元测试

1. 简介

工厂在组装一台电视机之前,会对每个元器件都进行测试,这就是单元测试;编写了一个函数,总要执行一下,验证功能是否正常,有时还要输出些数据,这也是单元测试。把这种单元测试称为临时单元测试。只进行了临时单元测试的软件,针对代码的测试很

不完整,代码覆盖率不足70%,未覆盖的代码可能遗留大量细小的错误,这些错误还会互相影响。当bug暴露出来的时候难于调试,大幅度提高了后期测试和维护成本,也降低了开发商的竞争力。因此,进行充分的单元测试,是提高软件质量,降低开发成本的必由之路。

单元测试是开发者编写的一小段代码,用于检验被测代码的一个很小的、很明确的功能是否正常。通常,一个单元测试是用于判断某个特定条件或场景下某个特定函数的行为。例如,把一个很大的值放入一个有序表中去,然后确认该值出现在有序表的尾部。

单元测试是由程序员自己来完成,最终受益的也是程序员自己。程序员有责任编写功能代码,同时也就有责任为自己的代码编写单元测试。执行单元测试,就是为了证明这段代码的行为和预期的一致。

单元测试是在软件开发过程中进行的最低级别的测试活动,其测试的对象是软件设计的最小单位。在传统的结构化编程语言中(如C语言),单元测试的对象一般是函数或子程序。在面向对象的语言(如C++)中,单元测试的对象可以是类,也可以是类的成员函数。对Ada语言而言,单元测试可以在独立的过程和函数上进行,也可以在Ada包的级别上进行。单元测试的原则同样也可以扩展到第四代语言(4GL)中,这时单元一般定义为一个菜单或显示界面。

单元测试又称为模块测试。模块并没有严格的定义,不过按照一般的理解,模块应该具有以下的一些基本属性:

- 名字;
- 明确规定的功能;
- 内部使用的数据或称局部数据;
- 与其他模块或外界的数据联系;
- 实现其特定功能的算法;
- 可被其上层模块调用,也可调用其下属模块进行协同工作。

单元测试的目的是要检测程序模块中有无故障存在,也就是说,一开始并不是把程序作为一个整体来测试,而是首先集中注意力来测试程序中较小的结构块,以便发现并纠正模块内部的故障。单元测试还提供了同时测试多个模块的良机,从而在测试过程中引入了并行性。下面主要来说明单元测试的任务和方法。

2. 单元测试的任务

单元测试针对每个程序模块进行,解决以下五个方面的问题。

1)模块接口测试

模块接口测试是单元测试的基础。通过对被测模块的数据进行测试,检查进出模块的数据是否正确。只有在数据能够正确地进入、流出的前提下,其他测试才有意义。因此,必须对模块接口,包括参数表、调用子模块的参数、全局变量、文件I/O操作进行测试,模块接口测试应该考虑下列一些因素:

- 模块接受输入的实际参数个数与形参的个数是否相同。
- 模块输入实际参数的属性与形参的类型是否匹配。

- 模块输入实际参数的使用单位与形参的使用单位是否一致。
- 调用其他模块时，所传送的实际参数个数与被调用模块形参的个数是否相同。
- 调用其他模块时，所传送的实际参数的属性与被调用模块形参的属性是否匹配。
- 调用其他模块时，所传送的实际参数的使用单位与被调用模块形参的使用单位是否一致。
- 调用内部函数时，所使用参数的个数、属性和次序是否正确。
- 在模块有多个入口的情况下，是否有与当前入口无关的参数引用。
- 是否修改了只读型的形参。
- 各模块对全局变量的定义是否一致。
- 是否把某些常数当做变量来传递。

如果模块涉及了外部的输入/输出，还应该考虑下列因素：

- 文件属性是否正确。
- OPEN/CLOSE 语句是否正确。
- 格式说明与输入/输出语句是否匹配。
- 缓冲区大小与记录长度是否匹配。
- 文件使用前是否已经打开。
- 在结束文件处理时是否关闭了文件。
- 输入/输出错误是否检查并做了处理。
- 输出信息中是否有文字性错误。

2）模块局部数据结构测试

局部数据结构检查临时存放在模块内的数据在程序执行过程中是否正确、完整，包括内部数据的内容、形式以及相互之间的关系。局部数据结构往往是故障的根源，应注意发现下面的几类错误：

- 不正确或不相容的类型说明。
- 不正确的初始化或默认值。
- 不正确的变量名。
- 数据下溢、数据上溢或地址异常等。

除局部数据结构外，单元测试还应检测全局数据对模块的影响。

3）错误处理检测

程序运行出现异常并不奇怪，良好的设计应该预先估计到各种可能的出错情况，并给出相应的处理措施，使用户遇到这些情况时不至于束手无策。检验程序错误处理也是单元测试的一个任务。检测模块错误处理是否有效，检查模块的错误处理功能是否包含有错误或缺陷。例如，是否拒绝不合理的输入；出错描述是否难以理解、是否对错误定位有误、是否出错原因报告有误、是否对错误条件的处理不正确；在对错误处理之前，错误条件是否已经引起系统的干预等。

对于可能出现的错误处理，应着重检查以下几种情况：

- 运行发生错误的描述是否难以理解。
- 所报告的错误与实际遇到的错误是否一致。

- 出错后，是否尚未进行出错处理便引入系统干预。
- 异常处理是否得当。
- 提供的错误信息不足，以致无法找到错误的原因。
- 错误描述中是否提供了足够的错误定位信息。

4）路径测试

检测模块运行能否满足特定的逻辑覆盖。单元测试应对模块中的每一条独立路径进行测试，以检测因计算错误、比较错误和不适当的控制流或判定而造成的故障。常见的计算错误包括：

- 误用或用错了算术优先级。
- 混合类型运算。
- 初始化错误。
- 计算精度不够。
- 表达式中符号表示错误。

比较判断常与控制流紧密相关，比较错误势必导致控制流错误，因此单元测试还应致力于发现以下错误：

- 不同数据类型的数据进行比较。
- 错误地使用逻辑运算符或优先级。
- 本应相等的数据由于精确度原因而不相等。
- 不正确的判定或不正确的变量。
- 循环终止不正确或循环不终止。
- 当遇到分支循环时不能退出。
- 错误地修改了循环控制变量。

5）模块边界条件测试

在为限制数据处理而设置的边界处，测试模块是否能够正常工作。模块边界测试是单元测试的一个关键任务，很可能发现新的软件故障。实践表明，边界是特别容易出现故障的地方。一些可能与边界有关的数据类型有数值、速度、字符、地址、位置、尺寸、数量等。另外，考虑这些边界的第一个、最后一个、最小值、最大值、最长、最短、最高、最低、相邻、最远等特征。

在边界条件测试中，应设计测试用例检查以下情况：

- 在 n 次循环的第 0 次，第 1 次，第 n 次是否有错误。
- 运算或判断中取最大值、最小值时是否有错误。
- 数据流、控制流中刚好等于、大于、小于确定的比较值时是否有错误。

3. 单元测试方法

单元测试一般在编码之后进行。在源代码编制完成，经过评审和验证，确认没有语法错误之后，开始设计单元测试用例。

由于模块并不是独立的程序，每个模块在整个软件中并不是孤立的，在对每个模块进行单元测试时，需要考虑它和周围模块之间的相互联系。为模拟这一联系，在进行单

元测试时,必须设置若干个辅助测试模块,这些辅助模块分为两种:

(1) 驱动模块。相当于被测模块的主程序,用以模拟被测模块的上级模块,用于接收测试数据,并把这些数据传送给被测模块,启动被测模块,最后输出实测结果。

(2) 桩模块。相当于被测模块调用的子模块,用以模拟被测模块的下级模块。桩模块一般只进行很少的数据处理,不需要把子模块的所有功能都带进来,但不允许什么事情都不做。

被测模块与其相关的驱动模块和桩模块共同构成了一个"测试环境",如图3-6所示。图中设置了一个驱动模块和4个桩模块。驱动模块在单元测试中接受测试数据并将这些数据传递到被测试模块,启动被测模块,并打印出相应的结果。桩模块则由被测模块调用,它们仅作少量的数据处理,例如打印入口和返回,以便于检验被测模块与其下级模块之间的接口。

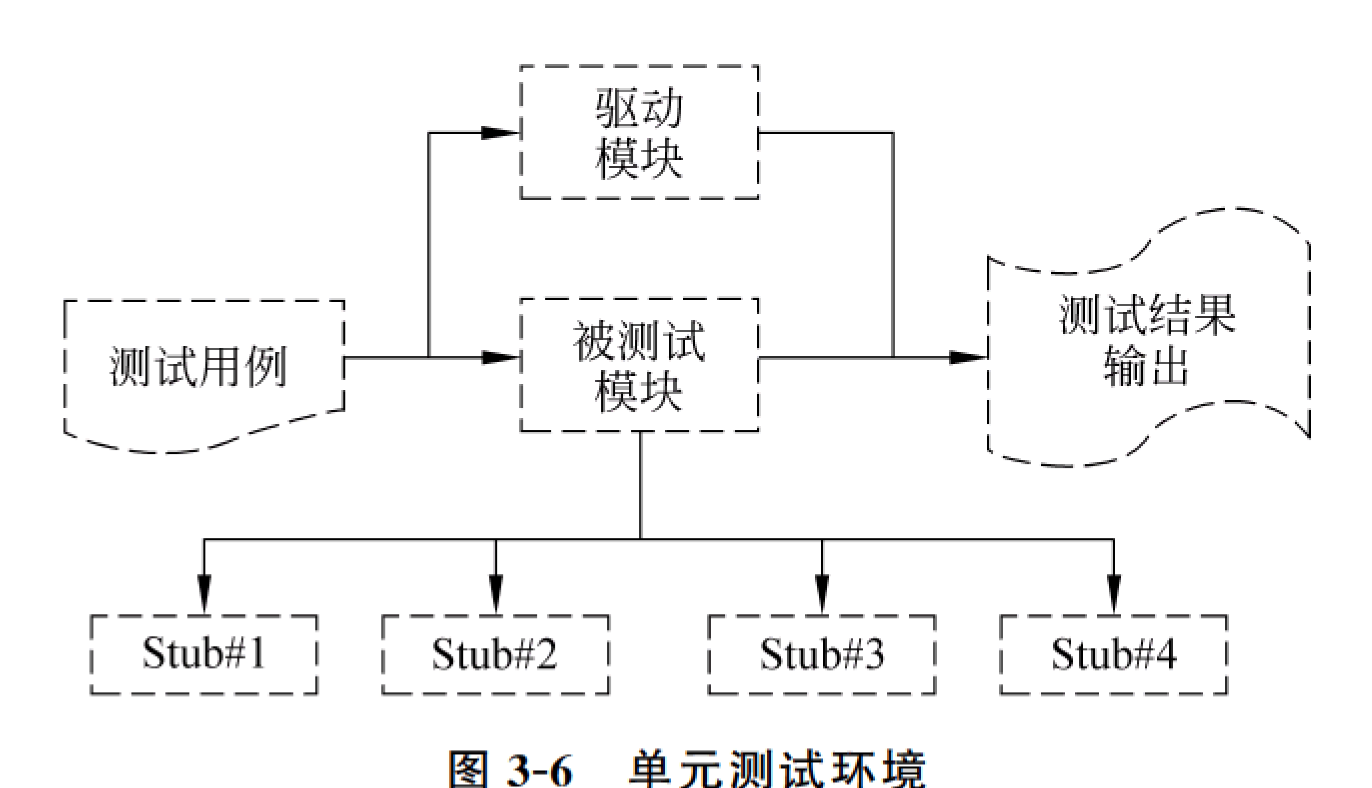


图3-6 单元测试环境

设计驱动模块和桩模块是一项额外的工作。虽然在单元测试中必须编写这些辅助模块,但它们却不是软件产品的组成部分。如果驱动模块和桩模块比较简单,实际开销相对低些。遗憾的是,有时仅用简单的驱动模块和桩模块并不能完成某些模块的测试任务,特别是桩模块,不能只简单地给出"曾经进入"的信息。为了能够正确地测试软件,可能需要模拟实际子模块的功能,建立这样的桩模块就比较困难了。

例3-1 单元测试实例。

某公司销售系统中关于数控机床的价格计算说明有:

(1) 数控机床价格的起点是:基准价(baseprice)减去折扣(discount)。其中,基准价是数控机床的基本价格,折扣是公司给予的在基准价基础上的价格折扣。

(2) 增加附加设备的附加价格(extraprice)。如果选择了3个或更多的附加设备,这些附加设备可以有10%的折扣。如果选择了5个或更多的附加设备,这些附加设备可以有15%的折扣。

(3) 销售商提供的折扣只针对基准价,这些折扣不能相互叠加。

下面的Java方法用于计算总价格:

```
double calculate_price(double baseprice,double extraprice,int extras,double
discount)
{
```

```
    double addon_discount,result;
    if (extras>=3) addon_discount=10;
    else if (extras>=5) addon_discount=15;
    else addon_discount=0;
    if (discount>addon_discount) addon_discount=discount;
    result=baseprice/100.0* (100-discount)+extraprice/100.0* (100-
    addon_discount);
    return result;
}
```

事实上，这个程序有一个缺陷：对于 extras 不低于 5 的情况永远都不会被执行。测试人员使用相应的类的接口测试这个价格计算器，用适当的参数和数据调用 calculate_price()，然后读取并记录函数调用的返回值。为了实现这样的测试，需要测试驱动模块，用来调用被测试组件(例如 calculate_price())，并获得测试对象返回值。

对于测试对象 calculate_price()，可以设计一个非常简单的测试模块：

```
boolean test_calculate_price()
{
    double price;
    boolean test_ok=true;
    //testcase 01
    price=calculate_price(10000.0,1000.0,3,0);
    test_ok=test_ok&&(Math.abs(price-12900.0)<0.001);          //浮点数不能直接比较
    //testcase 02
    price=calculate_price(25500.0,6000.0,6,0);
    test_ok=test_ok&&(Math.abs(price-34050.0)<0.001);
    //testcase...
    //test result
    return test_ok;
}
```

上面是一个简单的测试模块，可以对它进行一些有效的扩展。例如，开发一个工具来记录测试数据和测试结果，包括测试日期和时间，或者增加一个可以从文件或者数据库中读取测试用例的函数[7]。

在实际软件开发工作中，单元测试和代码编写所花费的精力大致相同。经验表明：单元测试可以发现很多的软件故障，并且修改它们的成本也很低。在软件开发的后期，发现并修复软件故障将变得更加困难，将花费大量的时间和费用。因此，有效的单元测试是保证全局质量的一个重要部分。在经过测试单元后，系统集成过程将会大大地简化，开发人员可以将精力集中在单元之间的交互作用和全局的功能实现上，而不是陷入充满故障的单元之中不能自拔。

3.5.2 集成测试

1. 简介

集成测试，也称组装测试或联合测试。在单元测试的基础上，将所有模块按照设计

要求(如结构图等)组装成为子系统或系统,进行集成测试。实践表明,一些模块虽然能够单独地工作,但并不能保证连接起来也能正常的工作。程序在某些局部反映不出来的问题,在全局上很可能暴露出来,影响功能的实现。可能的原因有:

- 模块相互调用时引入了新的问题。
- 几个子功能组合起来不能实现主功能。
- 误差不断积累达到不可接受的程度。
- 全局数据结构出现错误。

2. 集成测试任务

集成测试的主要任务是要求软件系统符合实际软件结构,发现与接口有关的各种错误。集成测试的主要任务是解决以下五个方面的测试问题。

(1) 将各模块连接起来,检查模块相互调用时,数据经过接口是否丢失。

(2) 将各个子功能组合起来,检查能否达到预期要求的各项功能。

(3) 一个模块的功能是否会对另一个模块的功能产生不利的影响。

(4) 全局数据结构是否有问题,会不会被异常修改。

(5) 单个模块的误差积累起来,是否被放大,从而达到不可接受的程度。

3. 集成测试步骤

执行集成测试应遵循下面的步骤。

① 确认组成一个完整系统的各模块之间的关系。

② 评审模块之间的交互和通信需求,确认出模块间的接口。

③ 使用上述信息产生一套测试用例。

④ 采用增量式测试,依次将模块加入到系统,并测试新合并后的系统。

4. 集成测试方法

集成测试是在每个模块完成单元测试以后,按设计要求把通过单元测试的各模块组装在一起,检测与接口有关的各种故障。那么,如何组织集成测试呢?是独立地测试每个模块,然后再把它们组合成一个整体进行测试,还是先把下一个待测模块组合到已测试过的那些模块上,再进行测试,逐步完成集成。前一种方法称为非增式集成测试法;后一种方法叫做增式集成测试法。

1) 非增式集成测试法

非增式测试是采用一步到位的方法来构造测试。对所有模块进行独立的单元测试后,按照程序结构图将各模块连接起来,把连接后的程序当做一个整体进行测试。

非增式测试的缺点:当一次集成的模块较多时,非增式测试容易出现混乱。因为测试时可能发现了许多故障,为每一个故障定位和纠正非常困难,并且在修正一个故障的同时,可能又引入了新的故障,新旧故障混杂,很难判定出错的具体原因和位置。

例 3-2　集成测试实例。

图 3-7 所示的程序中，7 个矩形分别表示程序的 7 个模块（子程序或者过程），模块之间的连线表示程序的控制层次——模块 M1 调用模块 M2、M3 和 M4，模块 M2 调用模块 M5 和 M6，模块 M4 调用模块 M7。

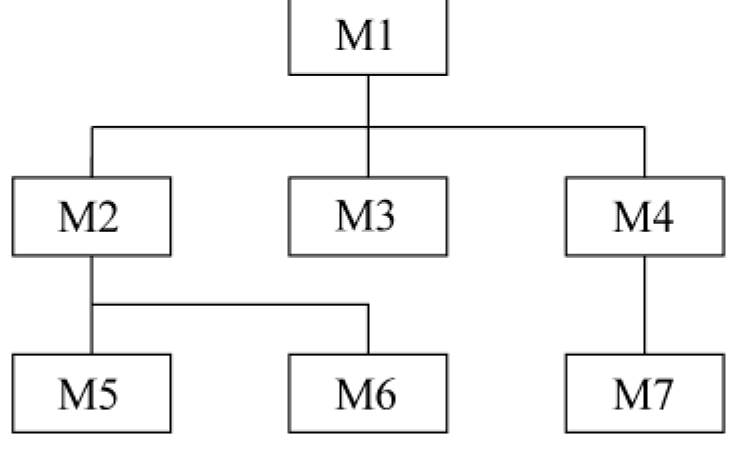


图 3-7　7 个模块组成的程序简图

非增式测试法的集成过程是：先对 7 个模块中的每一个进行单元测试。可以同时测试或是逐个地测试各个模块，这主要由测试环境（如所用计算机是交互式还是批处理式）和参加测试的人数等情况来决定；然后，在此基础上按程序结构图将各模块连接起来，把连接后的程序当做一个整体进行测试。

2）增式集成测试法

增式集成测试法不是孤立地测试每一个模块，而是一开始就把待测模块与已测试过的模块集合连接起来。增式集成测试可以分为自顶向下增式集成测试、自底向上增式集成测试和三明治集成测试。

自底向上增式集成测试从程序底部开始测试。仍以图 3-7 为例，可以先由 4 个人平行地测试或顺序地测试模块 M3、M5、M6 和 M7，然后测试模块 M2 和 M4，它们不是孤立地测试，而是把模块 M2 连在模块 M5 和 M6 上，模块 M4 连在模块 M7 上。自底向上增式集成测试过程，就是不断地把待测模块连接到已测模块集（或其子集）上，对待测模块进行测试，直到最后一个模块 M1 测试完毕。

自顶向下增式集成测试按结构图自上而下逐步集成并测试。模块集成的顺序是首先集成主控模块（主程序）；然后按照软件控制层次结构向下进行集成。自顶向下的集成方式可以采用深度优先策略和广度优先策略。由图 3-7 可知，深度优先顺序为 M1→M2→M5→M6→M3→M4→M7；广度顺序为：M1→M2→M3→M4→M5→M6→M7。

集成测试的整个过程由下列三个步骤完成。

① 以主控模块作为测试模块兼驱动模块，而所有直属于主模块的下属模块全部用桩模块替换，并对主模块进行测试。

② 依照所选用的模块集成策略（深度优先和广度优先），用实际模块替换相应桩模块，再用桩模块代替它们的直接下属模块，从而与已经测试的模块或子系统组装成新的子系统。

③ 在每个模块被集成时，都必须立即进行一遍测试。回到步骤②重复进行，直到整个系统结构被集成完成[8]。

最后需要说明的是，在软件集成阶段，测试的复杂程度远远超过单元测试的复杂程度。

3.5.3　确认测试

集成测试完成以后，分散开发的模块被连接起来，构成了一个完整的程序。其中各模块之间存在的种种问题已被消除，于是进入了确认测试阶段。

确认测试又称有效性测试，是对照软件需求规格说明，对软件产品进行评估以确定

其是否满足软件需求的过程。例如,编写出的程序相对于软件需求规格说明是否符合要求,程序输出的信息是否用户所要求的信息,程序在整个系统的环境中能否正确稳定地运行,对软件需求满足程度的评价等。

在软件开发过程中或完成以后,为了对它在功能、性能、接口以及限制条件等方面做出切实的评价,就应进行确认测试。实现软件确认要通过一系列黑盒测试。在开发的初期,软件需求规格说明中可能明确规定了确认标准,但在测试阶段需要更详细、更具体地在测试规格说明中加以体现。确认测试同样需要制定测试计划和过程。测试计划应规定测试的种类和测试进度。测试过程则定义一些特殊的测试用例,旨在说明软件与需求是否一致。无论是计划还是过程,都应该着重考虑软件是否满足合同规定的所有功能和性能,文档资料是否完整、准确,人机界面和其他方面(如可移植性、兼容性、错误恢复能力和可维护性等)是否令用户满意。

确认测试的另一个重要环节是配置审查工作。配置审查的文件资料包括用户手册、操作手册和设计资料。其目的在于保证软件配置齐全、分类有序,并且包括软件维护所必需的细节。

经过确认测试,应该为已开发的软件做出结论性的评价,这无非存在两种情况:

(1) 经过检验,软件功能、性能及其他方面的要求都已满足软件需求规格说明的规定,是一个合格的软件。

(2) 经过检验,发现与软件需求规格说明有相当的偏离,得到了一个缺陷清单,这就需要开发部门和用户进行协商,找出解决的办法。

3.5.4 系统测试

1. 简介

软件只是计算机系统的一个重要组成部分,软件开发完成以后,还应与系统中其他部分联合起来,进行一系列系统集成和测试,以保证系统各组成部分能够协调地工作。这里所说的系统组成部分除软件外,还包括计算机硬件及相关的外围设备、数据及采集和传输机构、计算机系统操作人员等。系统测试的目标不是要找出软件故障,而是要证明系统的性能。比如,确定系统是否满足其性能需求,确定系统的峰值负载条件及在此条件下程序能否在要求的时间间隔内处理要求的负载,确定系统使用资源(存储器、磁盘空间等)是否会超界,确定安装过程中是否会导致不正确的方式,确定系统或程序出现故障之后能否满足恢复性需求,确定系统是否满足可靠性要求等。

2. 系统测试前的准备工作

系统测试是检验软件的各种功能是否正常,并检验它的性能、强度、兼容性、使用性能、故障修复等一系列质量指标。因此,系统测试是一个庞大的工程,在测试之前要做如下的准备工作:

- 收集软件规格说明书,作为系统测试的依据。
- 收集各种软件说明书,以作为系统测试的参考。

- 仔细阅读软件测试计划书，尤其是独立的系统测试计划书，以作为系统测试的根据。
- 收集已编好的系统测试用例。

3. 系统测试技术和数据

系统测试完全采用黑盒测试技术，因为这时已不需要考虑组件模块的实现细节，而主要是根据需求分析时确定的标准检验软件是否满足功能、行为、性能和系统协调性等方面的要求。

系统测试一般要完成功能测试、性能测试、恢复测试、安全测试、强度测试以及其他限制条件的测试。

系统测试的数据须为真实数据，或者所选数据在代表性、精度性以及数据量等方面应尽可能地与真实数据相似。

进行系统测试的人必须善于从用户的角度考虑问题，最好能彻底了解用户的看法和环境及软件的使用。显然，最好的人选就是一个或多个用户。然而，一般的用户没有前面所说的各类测试的能力和专业知识，所以理想的系统测试小组应由几位职业的系统测试专家、一两名用户代表、一两名软件设计者或分析者组成。

3.5.5 验收测试

1. 简介

验收测试是部署软件之前的最后一个测试操作。其目的是确保软件准备就绪，并且可以让最终用户将其用于执行软件的既定功能和任务。

验收测试完全采用黑盒测试技术，主要是用户代表通过执行其在平常使用系统时的典型任务来测试软件系统，根据业务需求分析，检验软件是否满足功能、行为、性能和系统协调性等方面的要求。

2. α、β 测试

事实上，软件开发人员不可能完全预见用户实际使用程序的情况。例如，用户可能错误地理解命令或提供一些奇怪的数据组合，也可能对设计者自认明了的输出信息迷惑不解等。因此，软件是否真正满足最终用户的要求，应由用户进行一系列验收测试。验收测试可以是非正式的测试，也可以是有计划、有系统的测试。验收测试可能长达数周甚至数月，不断暴露错误，导致开发延期。一款软件产品，可能拥有众多用户，但不可能由每个用户验收。此时多采用称为 α、β 测试的过程，以期发现那些似乎只有最终用户才能发现的问题。α 测试是指软件开发公司组织内部人员模拟各类用户行为对即将面市的软件产品(称为 α 版本)进行测试，试图发现错误并修正。α 测试的关键在于尽可能逼真地模拟实际运行环境和用户对软件产品的操作并尽最大努力涵盖所有可能的用户操作方式。经过 α 测试调整的软件产品称为 β 版本。紧随其后的 β 测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用 β 版本，并要求用户报告异常情况、提出批

评意见。然后软件开发公司再对β版本进行改错和完善。一般包括功能度、安全可靠性、易用性、可扩充性、兼容性、效率、资源占用率、用户文档八个方面[6]。

3.5.6 回归测试

回归测试是指软件系统被修改或扩充(如系统功能增强或升级)后重新进行的测试，是为了保证对软件所做的修改没有引入新的错误而重复进行的测试。为了防止软件的变更产生无法预料的副作用，不仅要对内容进行测试，还要重复进行过去已经进行过的测试，以证明修改没有引起未曾预料的后果，或证明修改后的软件仍能满足具体的需求。

在理想的测试环境中，程序每改变一次，测试人员都重新执行回归测试，一方面来验证新增加或修改功能的正确性；另一方面以确定是否在实现新功能的过程中引入了缺陷。

回归测试特别适用于较高阶段的测试过程，回归测试一般多在系统测试和验收测试环境下进行，以确保整个软件系统新的构造或新的版本仍然运行正确，或者确保软件系统的现有业务功能完好无损。

回归测试一般采用黑盒测试技术来测试软件的高级需求，而无须考虑软件的实现细节，也可能采用一些非功能测试来检查系统的增强或扩展是否影响了系统的性能特性，以及与其他系统间的互操作性和兼容性问题。

第 4 章



bug 跟踪管理

如前所述，人们习惯用“软件缺陷（bug）”描述软件存在的各种问题。本章将主要介绍有关软件缺陷的定义、产生的原因及缺陷如何穿透测试，还包括缺陷的分类和生命周期。

4.1 软件缺陷的定义

所谓软件缺陷，指的是那些导致系统或部件不能实现其功能的缺陷。1.2.2 节已经对软件缺陷的定义进行描述，本章节不再重复。

软件未实现产品说明书要求的功能，或者出现了产品说明书指明不会出现的错误，没有实现产品说明书虽未明确提及但应该实现的目标。甚至最后的软件难以理解、不宜使用、运行缓慢等都可以称作是软件缺陷。缺陷的产生，可能引起系统不能正常运行或某些功能失效。因此，弄明白产生缺陷的原因，并且有效、准确的定义和描述软件缺陷，可以使得软件缺陷得以快速修复，从而节约软件测试项目的成本和资源，提高产品的质量。

4.2 产生缺陷的原因

图形用户界面（GUI）、B/S 结构、面向对象设计、分布式运算、底层通信协议、超大型关系型数据库以及庞大的系统规模，使得软件及系统的复杂性呈指数增长，不具备软件开发经验的人很难理解和掌握这些概念。总体上来说，产生缺陷的原因主要有以下几个方面：

1. 交流不良或相互误解

大多数的项目开发人员存在以下几种性格特点阻碍系统交流。

（1）羞涩。多数项目开发人员在跟客户交流时不能很好的表达自己的观点，或者跟客户交流时总是用很小的声音说明自己的观点，或者静静地坐在会议室的角落，无法参与到别人的激烈讨论中。

（2）胆怯。项目参与人员缺乏对客户的了解，造成盲目跟从心理。交流时不敢反驳或者提出相反的意见。

（3）依赖。部分项目参与人员在交流时，只做好自身的笔记工作，将主要责任寄托在周围其他合作开发人员上。

（4）轻视。拥有专业知识的项目人员不重视客户的意见，或者认为客户所说的毫无科学根据，无法实现。

（5）健忘。在会议上对讨论到的主要问题不做笔记，结果在实际的设计或者开发过程中遗忘了部分要点和注意事项。

（6）误解。在复杂的项目里，每个项目开发人员的认知层面、各自拥有的知识、处事原则各不相同，难免会产生这种情况。可以通过相互培训及有效的交流来避免这种情况的发生。

2. 软件开发过程的复杂性

（1）软件需求定义难以做到清清楚楚，导致设计目标偏离客户的需求，从而引起功能或产品特性上的缺陷。

（2）软件系统结构非常复杂，而又无法构造一个完美的层次结构或组件结构，结果将导致意想不到的问题。

（3）新技术的采用，可能涉及技术或系统兼容性的问题，而事先没有考虑到。

（4）对程序逻辑路径或数据范围的边界考虑不周全，容易在边界条件上出错，或者超出边界条件后又缺少保护导致出错。

（5）没有考虑或处理好系统崩溃后的自我恢复、故障转移或数据的异地备份等情况，从而存在系统安全性、可靠性的隐患。

3. 团队工作的问题

（1）沟通不够、不流畅，导致不同阶段、不同团队的开发人员对问题的理解不一致。

（2）项目组成员技术水平参差不齐，或者新员工较多或培训不够等，也容易引起问题。

（3）程序员由于过度疲劳、不守规矩、心不在焉等问题造成开发过程中问题的出现。

4. 需求变化

需求的变化是复杂的，其影响也是多方面的。客户并不了解需求变化所带来的后果，并且只需要看到变化，却从不考虑变化所需的额外工作时间。需求变化的后果可能会造成重新设计或者日程调整，已完成工作重做或者被完全抛弃，整个项目环境可能要因此改变等。频繁而小的变化或者几次重大的变化，项目各部分之间已知或者未知的依赖关系就会相互影响，从而导致更多问题的出现。需求变化增加了项目操作的复杂性，产生了大量不确定因素，并且还可能打击参与人员的工作积极性。一个需求变化频繁的项目或者产品不具备任何的测试价值。

5. 时间压力

时间是一种宝贵的资源，这和某位将军说的一样：“我们的奢侈品是伏尔加，是鱼子酱，却不是时间”。既然是一种宝贵资源就应该珍惜并且合理加以运用，所有软件项目时间都需要被精确估算。但项目进度也掺杂不稳定的时间因素，由此会造成缺陷的产生。

6. 文档贫乏

贫乏或者差劲的文档使得代码维护和修改变得异常艰辛，其结果是带来许多错误。区分职业人员的方法并不是看他有几年的编码经验，而在于其是否有良好的“先文档编写后实现”的习惯。

7. 开发工具

总是希望通过更加先进的工具来避免 bug 的出现，这就患上了典型的“银弹综合症”。开发工具可能使我们摆脱某些问题的出现，并且提高工作效率。但实际上，现代的开发工具对整个软件质量尤其是可靠性并没有什么重大的影响。

4.3 缺陷如何穿透测试

软件生产过程中，经过多次测试被排查出来的 bug 统称为 Public bug。它们被项目组内部甚至管理层所认知（对于使用产品的客户来说是保密的）。客户的使用过程中被发现的 bug 统称为 Private bug。

bug 的产生是不可避免的。即使最优秀的开发过程都会产生 bug。bug 产生于开发过程的各个阶段，包括需求、分析设计、实施、测试、部署等阶段。可能有许多时间和机会去发现它，如果错过了就应该把这个结果归罪于测试的不正确和测试的不完全。测试并不是软件开发过程中独立的部分，它应该由项目管理者、设计人员、开发人员、测试人员一起来完成。缺陷穿过测试的原因主要有以下几点。

1. 规范过程的执行代价太大

正规的软件公司会引入 QA，对项目整个过程进行全方位的质量保证工作。但是执行 QA 需要调用很多的资源，就需要高水平的分析员加入。在设计和实现阶段，随着大量审查工作的介入，所有该阶段的参与人员都要付出更多的时间和精力来参与。并且这些形式的检查、审查和测试延长了整个项目的开发过程，这些附加的工作时间都会直接变成附加费用，大大增加了整个项目的造价。

2. 市场的决策

即使测试人员发现了产品中的 bug，但是公司会觉得修复 bug 将延长整个产品的发布时间，有可能错过销售的旺季，并且会打乱整个公司针对该产品的销售计划。在确认产品中的 bug 不是非常严重的情况下，软件依然可以面向销售。如果是面向航天、医疗、

股票交割的管理控制软件系统,带有 bug 的软件发布的后果是非常严重的,但对于某些行业该做法是可行的。

3. 时间过于紧迫

测试要花费大量的时间,至今尚未有一种自动化的测试工具能够全面和高效率地测试一套软件产品。如果是 QA 的工作,那就要占用更多的项目时间。测试工作和编码工作同时进行,并且在时间紧迫的时候,大多数测试员只是选择明显的几条程序路径测试或者输入不完整的测试数据,这些都造成了大量的隐藏 bug。

4. 缺少现场证据

测试人员在发现缺陷的同时不知道如何去有效地体现。如果软件存在一个严重的 bug,但是测试人员描述不清,开发人员看不明白如何重现,那么将严重地威胁到系统的安全。

bug 的可重现性与导致 bug 出现的原因有着密切的联系。有时候 bug 出现的状况会误导测试人员,如运行软件导致 Windows 系统出现了“非法操作”的异常,根据以往的经验可以判断是系统内存被非法访问引起的,而实际上可能是软件中某个函数除零导致内存溢出引起的。

bug 的可重现性体现了测试人员对软件系统的熟悉程度,同时也体现在操作的顺序上。很多测试人员并没有严谨的工作方式,总是在很多步操作后碰到了 bug,才会提笔慢慢回忆以前的操作,写出当前的出错状况。这样的工作方式本身就存在着大量的不可靠因素,导致 bug 的不可重现。

5. 人员过于自信

开发人员是非常不诚恳的测试人员,他们总是简单地认为,“我做的肯定没问题”或者“不可能呀,它在我的机器上跑得好好的”。有的时候项目管理者也很自负,过于相信团队成员的表现,而不去理会测试人员或者客户的抱怨。

同时,开发人员过于满足系统的各方面功能,也是自信的一种表现。当人感到满足的时候就不会质疑任何假设,并且把所有假设的输出都设为真。没有详细的测试计划,没有严谨的测试行为,不再关注每个细小的环节,导致 bug 没有被发现。

6. 模糊的 bug 提交

开发人员需要知道要实现什么,怎样把实现结果以更简练的方式书写出来。测试人员需要知道要测试什么,怎样把测试结果以更清楚的方式书写出来。图 4-1 和图 4-2 是两份不同的测试报告。

图 4-1 所示的 bug 报告很糟糕,至少属于三无报告:无详细错误说明、无详细出错描述、无详细错误跟踪。图 4-2 所展示的报告相对比较完善,比较详尽。由图可以看出 bug 的出处、严重性、关键字、责任人等,那么开发人员可以很快定位和重现 bug。



图 4-1 一份不详尽的缺陷报告

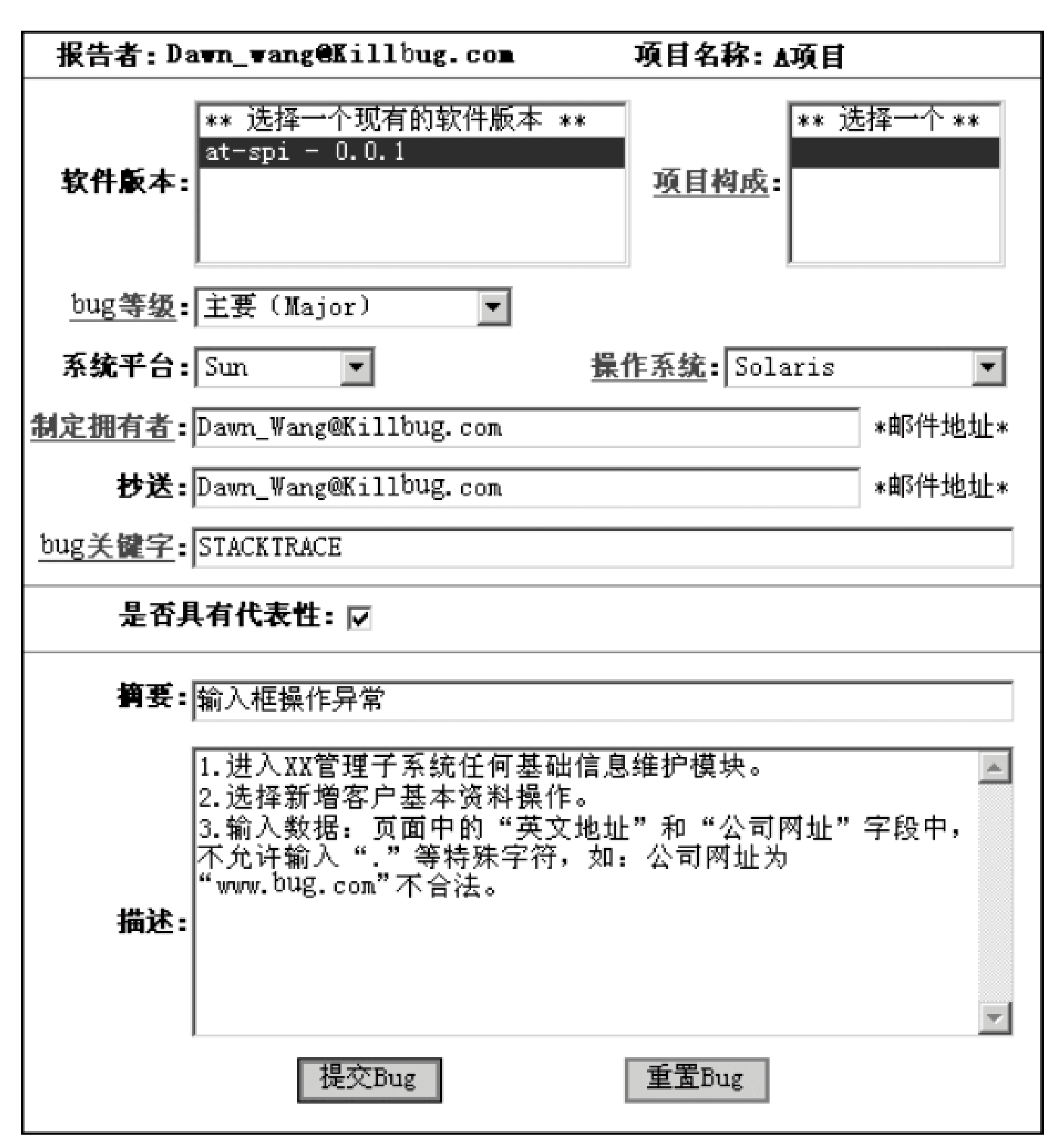


图 4-2 一份定位精确的错误报告

7. 测试环境的影响

一个好的测试环境需要具备三个要素：熟练使用工具的测试人员、多种不同配置的系统和好的测试模型。如果缺少必要的测试工具和设备，那么将很难完成测试。如在一个比较大型的网站中，如果测试人员没有有效的测试工具或者必要的硬件设备，那么就很难去模拟、再现系统负载的环境，也就无法测试系统在正常负载情况下的性能；如果缺少必要的系统配置，那么测试将很难进行。如基于 Java 开发的程序，同时还和某种硬件有所关联，如果缺少正确的系统配置，将无法在多种操作系统上去验证它的正确性和稳定性，甚至会得出错误的分析判断；如果缺少必要的测试模型，那么将会导致测试混乱，产生更多的 bug。好的测试模型不仅可以减少更多的 bug，而且可以发现更多潜在的 bug。但是好的测试模型并非一系列测试方法的组合，它更大的用处在于和历史积累的 bug 记录作对比分析。

需要说明的一点是，每一个使用过一些软件的人都会对软件的工作方式有自己的意见和想法，要编写令所有用户都满意的软件是不可能的。要全面，最重要的是要客观评价，并非所有测试发现的缺陷都要修改。

4.4 缺陷的分类

不同的软件缺陷会产生不同的后果。在时间、人力、物力允许的情况下，应修复软件中存在的所有缺陷。但是，这只是软件开发的一种理想情况。一方面，绝大多数软件项目都是要求按期完成的，而且给予项目的资源有限（包括人力资源和软件生产资料）。只

有对软件缺陷进行区别对待,区分其严重等级和优先等级,将有限的资源充分利用,解决对软件产品质量最为关键的软件缺陷,才能生产出用户满意的软件;另一方面,软件缺陷的修复都需要付出代价。在软件生命周期的各个阶段,相同的软件缺陷的修复成本大不相同。随着时间的推移,修复软件缺陷的费用呈几何级增加。因此要尽可能早地发现缺陷并修复缺陷[9]。

对软件缺陷进行分类,分析产生各类缺陷的软件过程原因,总结在开发软件过程中不同软件缺陷出现的频度,制定对应的软件过程管理与技术的改进措施,是提高软件组织的生产能力和软件质量的重要手段。

1. Putnam 等[10]将软件缺陷分为六类:需求缺陷、设计缺陷、文档缺陷、算法缺陷、界面缺陷和性能缺陷。国家军用标准 GJB-437,根据军用软件错误的来源,将软件错误分为三类:

(1) 程序错误。运行程序与相应的文档不一致,而文档是正确的。

(2) 文档错误。运行程序与相应的文档不一致,而程序是正确的。

(3) 设计错误。虽然运行程序与相应的文档一致,但是存在设计缺陷,产生错误。

这两种分类方法可以分析软件缺陷的来源和出处,指明修复缺陷的努力方向,为软件开发过程各项活动的改进提供线索。该分类方法的显著特点是分类简单。因为分类方法简单,所以提供的缺陷相关信息对具体的缺陷修复工作的贡献有限。

2. Thayer 软件错误分类方法[11]是按错误性质分类。它利用测试人员在软件测试过程填写的问题报告和用户使用软件过程反馈的问题报告作为错误分类的信息。它包括16个大类,164个子类。具体大类如下:

- 计算错误;
- 逻辑错误;
- I/O 错误;
- 数据加工错误;
- 操作系统和支持软件错误;
- 配置错误;
- 接口错误;
- 用户需求改变;
- 预置数据库错误;
- 全局变量错误;
- 重复的错误;
- 文档错误;
- 需求实现错误;
- 不明性质错误;
- 人员操作错误;
- 问题。

该分类方法特别适用于指导开发人员的缺陷消除和软件改进工作。通过对错误进

行分类统计,可以了解错误分布状况,对错误集中的位置重点加以改进。该方法分类详细,适用面广,当然分类也较为复杂。但该分类方法没有考虑造成缺陷的过程原因,不适用于软件过程改进活动。

3. 缺陷正交分类(Orthogonal Defects Classification, ODC)[12]是 IBM 公司提出的缺陷分类方法。该分类方法提供一个从缺陷中提取关键信息的测量范例,用于评价软件开发过程,提出正确的过程改进方案。该分类方法用多个属性来描述缺陷特征。在 ODC 最新版本里,缺陷特征包括以下八个属性:发现缺陷的活动、缺陷影响、缺陷引发事件、缺陷载体、缺陷年龄、缺陷来源、缺陷类型和缺陷限定词。ODC 对八个属性分别进行了分类,其中缺陷类型被分为七大类:赋值、检验、算法、时序、接口、功能和关联。

分类过程分两步进行。第一步,缺陷打开时,导致缺陷暴露的环境和缺陷对用户可能的影响是易见的。此时可以确定缺陷的三个属性:发现缺陷的活动、缺陷引发事件和缺陷影响。第二步,缺陷修复关闭时,可以确定缺陷的其余五个属性:缺陷载体、缺陷类型、缺陷限定词、缺陷年龄和缺陷来源。这八个属性对于缺陷的消除和预防起到关键作用。

该分类方法分类细致,适用于缺陷的定位、排除、缺陷原因分析和缺陷预防活动。缺陷特征提供的丰富信息为缺陷的消除、预防和软件过程的改进创造了条件。IBM 的研究机构制定出该分类方法已经十多年,并一直在改进。近几年,业界开始研究并使用 ODC。其缺点在于分类复杂,难以把握分类标准,缺陷分析人员的主观意见会影响属性的确定。

4. 电气和电子工程师学会制定的软件异常分类标准(IEEE Standard Classification for Anomalies 1044—1993)[13]对软件异常进行了全面的分类。该标准描述了软件生命周期各个阶段发现的软件异常的处理过程。分类过程由识别、调查、行动计划和实施处理四个步骤组成,每一步骤包括三项活动:记录、分类和确定影响。异常的描述数据称为支持数据项。分类编码由两个字母和三个数字组成。如果需要进一步的分类,可以添加小数。例如 RR324,IV321.1 等。其中,RR 表示识别步骤;IV 表示调查步骤;AC 表示行动计划步骤;IM 表示确定影响活动;DP 表示实施处理步骤。分类过程的四个步骤都需要支持数据项。由于每个项目都有各自的支持数据项,该标准不强制规定支持数据项,但提供了各个步骤相关的建议支持数据项。强制分类建立通用的定义术语和概念,便于项目之间、商业环境之间、人员之间的交流沟通。可选分类提供对于特殊情况有用的额外的细节。在调查步骤,对实际原因、来源和类型进行了强制分类。其中调查步骤将异常类型分为逻辑问题、计算问题、接口/时序问题、数据处理问题、数据问题、文档问题、文档质量问题和强化问题八个大类,下面又分为数量不等的小类。分类细致深入,准确说明了异常的类型。

该分类方法提供一个统一的方法对软件和文档中发现的异常进行详细的分类,并提供异常的相关数据项帮助异常的识别和异常的跟踪活动。该分类标准具有较高的权威性,可针对实际的软件项目进行裁剪,灵活度高,应用面广,同时提供了丰富的缺陷信息,有助于进行缺陷原因分析。不足之处是没有考虑软件工程的过程缺陷,并且分类过程复杂。

除了上述四种缺陷分类方法外，还可以根据具体分类标准进行分类，如根据缺陷类型、缺陷严重程度、缺陷优先级、缺陷状态以及缺陷来源进行分类。其中缺陷状态、缺陷严重程度和缺陷优先级在接下来章节中会介绍。

根据缺陷类型进行分类，具体如表 4-1 所示。

表 4-1 根据缺陷类型进行分类

编号	缺陷类型	描述	子类型	
			编号	名称
01	功能问题 F-Function	影响了重要的特性、用户界面、产品接口、硬件结构接口和全局数据结构。并且设计文档需要正式的变更。如指针、循环、递归、功能等缺陷	0101	功能错误
			0102	功能缺失
			0103	功能超越
			0104	设计二义性
			0105	算法错误
02	接口问题 I-Interface	与其他组件、模块或设备驱动程序、调用参数、控制块或参数列表相互影响的缺陷	0201	模块间接口
			0202	模块内接口
			0203	公共数据使用
03	逻辑问题 L-Logic	需要进行逻辑分析，进行代码修改，如循环条件等	0301	分支不正确
			0302	重复的逻辑
			0303	忽略极端条件
			0304	不必要的功能
			0305	误解
			0306	条件测试错误
			0307	循环不正确
			0308	错误的变量检查
			0309	计算顺序错误
			0310	逻辑顺序错误
04	计算问题 C-Computation	等式、符号、操作符或操作书错误，精度不够、不适当的数据验证等缺陷	0401	等式错误
			0402	缺少运算符
			0403	错误的操作数
			0404	括号用法不正确
			0405	精度不够
			0406	舍入错误
			0407	符号错误

续表

编号	缺陷类型	描述	子类型	
			编号	名称
05	数据问题 A-Assignment	需要修改少量代码，如初始化或控制块。如声明、重复命名，范围、限定等缺陷	0501	初始化错误
			0502	存取错误
			0503	引用错误的变量
			0504	数组引用越界
			0505	不一致的子程序参数
			0506	数据单位不正确
			0507	数据维数不正确
			0508	变量类型不正确
			0509	数据范围不正确
			0510	操作符数据错误
			0511	变量定位错误
			0512	数据覆盖
			0513	外部数据错误
			0514	输出数据错误
			0515	输入数据错误
			0516	数据检验错误
06	用户界面问题 U-User Interface	人机交互特性：屏幕格式，确认用户输入，功能有效性，页面排版等方面的缺陷	0601	界面风格不统一
			0602	屏幕上的信息不可用
			0603	屏幕上的错误信息
			0604	界面功能布局和操作不合常规
07	文档问题 D-Documentation	影响发布和维护，包括注释等缺陷	0701	描述含糊
			0702	项描述不完整
			0703	项描述不正确
			0704	项缺少或多余
			0705	项不能验证
			0706	项不能完成
			0707	不符合标准
			0708	与需求不一致
			0709	文字排版错误
			0710	文档信息错误
			0711	注释缺陷

续表

编号	缺陷类型	描述	子类型	
			编号	名称
08	性能问题 P-Performance	不满足系统可测量的属性值，如：执行时间，事务处理速率等缺陷		
09	配置问题 B-Build/package/merge	由于配置库、变更管理或版本控制引起的错误	0901	配置管理问题
			0902	编译打包缺陷
			0903	变更缺陷
			0904	纠错缺陷
10	标准问题 N-Norms	不符合各种标准的要求，如编码标准、设计符号等缺陷	1001	不符合编码标准
			1002	不符合软件标准
			1003	不符合行业标准
11	环境问题 E-Enviroments	由于设计、编译和运行环境引发的问题	1101	设计、编译环境
			1102	运行环境
12	其他问题 O-Others	以上问题所不包含的其他问题		

根据缺陷来源分类，具体如表4-2所示。

表4-2 根据缺陷来源进行分类

序号	缺陷来源	描述
1	需求(Requirement)	由于需求分析不当所引起的缺陷
2	架构(ArchitecTrue)	由于构架的问题引起的缺陷
3	设计(Design)	由于设计的问题引起的缺陷
4	编码(Coding)	由于编码的问题引起的缺陷
5	测试(Test)	由于测试的问题引起的缺陷
6	集成(Integration)	由于集成的问题引起的缺陷

4.5 缺陷的生命周期

和软件开发生命周期一样，缺陷也是由一系列的阶段和活动组成的，同样具有生命周期。每一个bug都规定了七种生命状态，即bug初始状态(unconfirmed&new)、bug分配状态(assigned)、bug重新分配状态(reassigned)、bug修复状态(resolved&fixed)、bug验证状态(vertified&fixed)、bug重新打开状态(reopen)、bug关闭状态(closed&Fixed)。bug生命周期及各状态过程，如图4-3所示。

bug的流转状态关键字如表4-3所示。

表 4-3　bug 状态流转关键字含义

bug 的流转状态关键字	含　义
未确定的(unconfirmed)	最近才被发现,还没有人确认它是否真的存在。如果有别的测试人员碰到了同样的问题,就可以将该 bug 标志为 new,或者将该 bug 删除,或者做上 resolved 标记
新加入的(new)	最近被测试人员添加到 bug 列表中,已经被证实存在且必须修改的。即将被分配,如果分配了可以标志为 assigned,未分配则将保留 new 标志,或者将这个 bug 删除,或者做上 resolved 标记
确认分配的(assigned)	测试人员将 bug 的修复任务分配给具体的实现人员,如果 bug 不属于被分配实现人员的范围,可置为 reassigned,等待测试人员重新指定相关修改人员
重新分配的(reassigned)	该 bug 不属于被分配实现人员的范围,可置为 reassigned 等待测试人员重新指定相关修改人员
需要帮助的(needinfo)	测试人员无法对发现的 bug 进行精确定位或描述,需要相关实现人员协助,以更深刻地认识和修复这个 bug
修复的(resolved)	实现人员根据测试人员的意见进行 bug 修复
重新启用的(reopen)	当测试人员或实现人员发现某些 bug 具有关联性,即使该 bug 被正确修复了,也会被发送到起始状态等待回归再次确认
在验证的(verified)	实现人员根据测试人员对实现人员修复后的 bug 进行测试确认和验证,验证成功后并将该 bug 标记为 vertified&fixed 状态
安全关闭的(closed)	该 bug 已经被完全解决

图 4-3　bug 的生命流转图

从图中不难看出 bug 生命历程的五种典型过程:

- bug start → unconfirmed&new 状态 → assigned 状态 → reassigned 状态 → resolved&fixed 状态→vertified&fixed 状态→closed&fixed 状态。
- bug start→resolved&fixed 状态→vertified&fixed 状态→closed&fixed 状态。
- bug start→unconfirmed&new 状态→assigned 状态→reassigned 状态。
- bug start → unconfirmed&new 状态 → assigned 状态 → reassigned 状态 → resolved&fixed 状态→reopen 状态。
- bug start → unconfirmed&new 状态 → assigned 状态 → reassigned 状态 → resolved&fixed 状态→vertified&fixed 状态→reopen 状态。

下面将详细描述上述五种过程中是如何控制和转换的。

第一种过程,测试人员发现 bug 并且将其标记为 unconfirmed&new 状态,当测试人员在排除 bug 的登记错误后,将该 bug 置为 assigned 状态。实现人员接到该 bug 通告进行 bug 确认,如果被分配的 bug 不属于实现人员的范围,那么等待测试人员重新指定相

关修改人员,重新分配成功后并将该 bug 状态被置为 reassigned 状态。当实现人员修复 bug 后,该 bug 置为 resolved&fixed 状态。测试人员对实现人员修复后的 bug 进行测试确认和验证,验证成功后并将该 bug 标记为 vertified&fixed 状态。如果该 bug 被正确修复了,那么其状态被置为 closed&fixed 状态,同时意味着该 bug 的整个生命周期终结。

第二种过程,回归测试后,如果部分登记 bug 再次出现,测试人员可直接将已登记的 closed&fixed 状态的 bug 转入修复流程,等实现人员修复 bug 后将其置为 resolved&fixed 状态。测试人员对实现人员修复后的 bug 进行测试确认和验证,验证成功后并将该 bug 标记为 vertified&fixed 状态。如果该 bug 被正确修复了,那么其状态被置为 closed&fixed 状态,同时意味着该 bug 的整个生命周期终结。

第三种过程,测试人员发现 bug 并且将该 bug 标记为 unconfirmed&new 状态,下一步测试人员在排除 bug 的登记错误后,将该 bug 置为 assigned 状态。实现人员接到该 bug 通告进行 bug 确认,确认失败后该 bug 状态被置为 reassigned 状态并发送回 bug 起始阶段。

第四种过程,测试人员发现 bug 并且将该 bug 标记为 unconfirmed&new 状态,下一步测试人员在排除 bug 的登记错误后,将其置为 assigned 状态。实现人员接到该 bug 通告进行 bug 确认,如果被分配的 bug 不属于实现人员的范围,那么等待测试人员重新指定相关修改人员,重新分配成功后并将该 bug 状态被置为 reassigned 状态。当实现人员修复 bug 后该 bug 置为 resolved&fixed 状态,但是实现人员发现该 bug 与其他实现人员的 bug 有关联,可能导致本次修复无效,所以实现人员将该 bug 置为 reopen 状态发送回 bug 起始阶段。

第五种过程,测试人员发现 bug 并标记为 unconfirmed&new 状态,下一步测试人员在排除 bug 的登记错误后,将其置为 assigned 状态。实现人员接到该 bug 通告进行 bug 确认,确认成功后该 bug 状态被置为 reassigned 状态,当实现人员修复 bug 后该 bug 置为 resolved&fixed 状态。测试人员对实现人员修复后的 bug 进行测试确认和验证,验证成功后并将该 bug 标记为 vertified&fixed 状态。如果测试人员怀疑该 bug 并非真正修复,则将该 bug 置为 reopen 状态发送回 bug 起始阶段。

一般情况下,在测试初期被发现的 bug 数量会急剧上升,随着实现人员、测试人员的处理逐渐转少。当所有软件缺陷的状态都转为 closed&fixed 状态,且在一段时间内没有被打开,也没有新的 bug 发生时,测试可以结束或告一段落。

4.6 缺陷的严重程度和优先级

缺陷的严重程度和优先级是含义不同但相互联系密切的两个概念,它们从不同方面描述了软件缺陷对软件质量、用户、开发过程的影响程度和处理方式。一般来说,严重程度高的缺陷具有较高的优先级。严重程度高说明缺陷对软件造成的质量危害性大,需要优先处理。而严重性低的缺陷可能只是软件的瑕疵,可以稍后处理。但是优先级和严重程度并不总是一一对应的,也存在优先级低但严重程度高的缺陷,或者优先级高但严重程度低的软件缺陷。

下面分别介绍缺陷的严重程度等级和优先级[1]：

1. bug 的严重程度等级

(1) 危急的(critical)：能使不相关的系统内软件(或整个系统)损坏，或造成严重的信息遗失，或为安装该软件包的系统引入安全漏洞。

(2) 重大的(grave)：使该软件包无法或几乎不可用，或造成数据遗失，或引入一个允许侵入此软件包用户账号的安全漏洞。

(3) 严重的(serious)：该软件包违反了“必须”或“必要”的规定，或者是软件包维护人员和测试人员认为该软件包已不适合发布。

(4) 阻碍的(blocker)：阻碍了后面的操作，需要马上或者尽快排除。

(5) 重要的(important)：影响了软件包可用性，但不至于造成所有人都不可用。

(6) 常规的(normal)：是默认的，适用于大部分的错误。

(7) 轻微的(minor)：不至于影响软件的使用，而且应该很容易解决。

(8) 微不足道的(trivial)：无关紧要，多指外观 GUI 上的字符拼写错误，不影响整个项目。

2. bug 处理的优先等级

(1) 立刻修复(immediate)：已经阻碍了开发工作或者测试工作，需立刻修改。

(2) 马上修复(urgent)：阻碍了软件的部分应用，不修复将妨碍下面计划的实施。

(3) 尽快修复(high)：真实存在的并不是很严重，在版本发布之前修复。

(4) 正常修复(normal)：有充足的时间来修复，并且对现行系统的影响不大。

(5) 考虑修复(low)：不是关键的，时间允许的时候可以考虑修复。

正确区分和处理缺陷严重程度和优先级，是软件质量保证的重要环节。因此，正确处理和区分缺陷的严重程度和优先级是所有的软件开发和测试相关人员的重要职责，需要正确理解缺陷严重程度和优先级的含义，同时认识到这是保证软件质量的重要环节，应该引起足够的重视。将比较轻微的缺陷设置成高严重程度和高优先级的缺陷，夸大缺陷的严重程度，将影响软件质量的正确评估，耗费开发人员辨别和处理缺陷的时间；而将严重的缺陷报告成低严重程度和优先级的缺陷，这样会掩盖许多严重的缺陷。如果在项目或者软件发布前，发现还有很多由于不正确分配优先级造成的严重缺陷，将需要投入很多人力和时间进行修改，影响软件的正常发布。或者这些严重的缺陷成为漏网之鱼，随着软件一起发布出去，就会影响软件的质量并降低用户使用软件的信心。

修改软件缺陷并不是纯技术的问题，有时候需要考虑软件版本发布和质量风险等因素。下面是关于缺陷严重程度和优先级设置方面的一些建议：如果某个严重的缺陷只在非常极端的条件下产生，则可以将缺陷的优先级设置得比较低；如果修正一个软件缺陷需要重新修改软件的整体架构，可能会产生更多的潜在缺陷，而且市场要求尽快发布软件版本，那么即使这个缺陷严重程度很高，也需要仔细考虑是否需要修改；对于有些缺陷，可能它的严重程度很低，但其市场影响力较大，例如界面中公司名称或商标单词拼写

错误，则这个缺陷的优先级就很高，必须尽快进行修复。

4.7 缺陷的描述

软件缺陷的描述是软件缺陷报告中测试人员对问题陈述的一部分。软件缺陷的描述需要具备简单、准确以及专业等特点，否则会使信息含糊不清，可能会误导开发人员。以下是软件缺陷的有效描述规则：

(1) 单一准确。每个报告只针对一个缺陷，在一个报告中报告多个弊端常常会导致只有其中一个软件缺陷得到注意和修复。

(2) 再现性。提供该缺陷的精确步骤，使开发人员容易看懂，可以再现并修复缺陷。

(3) 完整统一。提供完整、前后统一的软件缺陷的修复步骤和信息，例如图片信息、日志文件等。

(4) 短小精简。通过使用关键词，可以使软件缺陷的标题描述短小简练，又能准确解释产生缺陷的现象。如“存货量低于0并没有发生错误提示报告”中的“存货量”，“低于0”是关键词。

(5) 特定条件。许多软件功能在通常情况下没有问题，而在特定情况下会存在缺陷，所以软件缺陷描述不要忽视这些看似细节但又必要的特定条件(如特定的操作系统、浏览器或者某种设置等)，能够提供帮助开发人员找到原因的线索。例如，网站在IE浏览器和Firefox浏览器之间的兼容性。

(6) 补充完善。从发现bug开始，测试人员的责任就是保证它被正确的报告，并且得到应有的重视。

(7) 不做评价。软件缺陷描述不要带有个人观点，不要对开发人员进行评价。软件缺陷报告是针对产品的。

缺陷描述分问题摘要和详细描述两大部分，以PHPWind Blog系统为例，测试中的缺陷描述如表4-4所示。

表4-4 缺陷描述示例

问题摘要	相对应的详细描述
“用户功能模块”中“用户注册”功能问题	在“用户注册”界面中，输入密码时输入特殊字符，用户提交时应当报错或提示：如果用户在输入密码时，密码中含有特殊字符，系统应当报错
“客户端设置子系统”模块中“日志管理”功能问题	在“日志管理-添加日志-添加附件”的功能中，“混排操作功能”选项，使用无效，不能进行图文混合排列
“用户设置”模块中“修改个人设置”功能问题	在“修改个人设置”界面中，应当添加在以下情况时的报错或提示：如果输入的新密码和确认密码不是数字与字母的组合，系统应当提示“密码包含不可接受字符，请使用英文和数字”；如果输入的新密码和确认密码少于6位时，系统应当提示“密码太少，请用6位以上”

提交一条缺陷后，最好能够再检查一遍缺陷格式是否有问题。常见的格式问题如下：

- 问题摘要中不能有句号。
- 问题摘要后不要有空格，直接填写内容。
- 问题摘要比较长时，可以用“，”分隔。
- 详细描述中序号后面“.”一定是半角，而不是全角符号，并且后面不要再有空格。
- 详细描述中分号一定要使用全角的分号。
- 详细描述中的“→”应统一。要在英文输入法的半角状态下输入箭头。
- 注意缺陷中不要出现错别字。

另外，缺陷描述常见的问题还有：

- 问题摘要过长，不够简练、准确。
- 问题摘要与详细描述的内容不一致。
- 详细描述不清楚，无法复现。
- 详细描述冗长，不宜于理解。

第 5 章



软件自动化测试基本理论

5.1 软件自动化测试基础

软件自动化测试(Automated Testing,AT)是相对手工测试而存在的,主要通过所开发的软件测试工具、脚本等来实现,具有良好的可操作性、可重复性和高效率等特点[5]。目前,自动化测试已经成为国内软件工程领域的一个重要课题。

5.1.1 自动化测试的定义

朱少民[14]将自动化测试定义为:一切可以由计算机系统自动完成的测试任务都已由计算机系统或软件工具、程序来承担并自动执行。它包含了下列三层含义:

(1)"一切",不仅包括测试执行的任务,即验证被测对象,而且还包括测试相关的其他任务,如环境安装、测试管理、缺陷管理、设置和维护等。

(2)"可以",系统不能自动完成测试中的某些任务,如脚本开发、测试用例设计等。类似这些需要创造性的任务依旧必须通过手工处理完成。

(3)即使由系统进行自动化测试,还少不了人的干预,包括事先安排自动化测试任务、测试结果分析、调试测试脚本等。

严格地说,自动化测试与测试自动化之间存在区别。自动化测试是对手工测试步骤进行模拟,通过执行脚本自动测试软件,并自动实施软件的单元测试、功能测试、负载测试及性能测试等。一方面,强调实际测试执行过程的改变,即测试工具自动执行的过程代替了传统的手工逐个运行测试用例的操作过程;另一方面,强调借助工具、策略等来完成测试的执行,即用工具来帮助或辅助测试,该过程可能是全自动的,也可能是半自动的。相比较而言,测试自动化则侧重自动化设计和实现测试的整个过程,即所有的测试任务全部都可以用计算机系统自动地完成,包括:环境的自动搭建及设置(如安装包上载到服务器);脚本(如根据 UML 状态图、时序图等生成可运行的测试脚本)的自动生成;测试数据(如负载测试所需的大量数据)的自动产生;操作步骤(含执行过程的控制)的自动执行;结果(如实际输出结果和预期输出结果)的自动分析;测试流程(如测试计划复审和批准、测试任务安排和执行、缺陷生命周期等)的自动处理及实现;测试报告自动生成等。

测试自动化意味着测试全过程的自动化和测试管理工作的完全自动化。然而，往往很难完全通过全自动化过程来完成一个完整的测试任务，换言之，自动化到不需要人工参与的程度是不现实的[14]。

但在一般情况下，“自动化测试”和“测试自动化”之间并无严格区分。本书定义自动化测试为：使用测试工具或其他手段代替或辅助人工来验证各种软件测试的需求，包括测试活动的管理与实施。其目的是通过较少的开销，得到更彻底的测试，以提高工作效率和产品质量，而不是为了自动化而自动化。妄图做到完全百分之百地自动化测试，不仅不现实，甚至可能会引起较大的代价，而且可能引起负面性，直接导致质量的降低。

自动化测试仅仅是测试工作的一部分，是对手工测试的一种补充。自动化测试和手工测试往往交织进行，相互补充。工具执行过程往往需要人工分析，手工测试时也可借助工具处理及显示某些数据、日志等信息。自动化测试绝不能完全代替手工测试，而应在尊重手工测试的基础上，尽量采用自动化测试，根据各自的特点充分发挥各自的优势，使手工测试和自动化测试实现完美结合，从而达到高效测试的目的。

5.1.2 自动化测试的对象

通常，试图实现所有测试工作自动化都不是很有成效，而且可能是不现实或不合理的，例如，人机界面的测试过程就不适合自动化，因为人是该测试的关键因素。一般来说，如果写一个自动化脚本需要的时间比写一个手工脚本需要的时间长，就需要估计使用自动化脚本所节省的时间及该脚本执行的次数，以确定该脚本是否适合自动化。如果某项测试预计只运行一次，那么，除非实现脚本自动化的成本非常低（如捕获回放），否则手工测试将更加有效。比较适合自动化测试的对象有[15]：

1. 重复的任务

重复的任务是自动化测试的主要候选对象，主要包括回归测试、烟雾测试、负载测试和性能测试等。其中，烟雾测试可以看作回归测试的一个子集，其运行次数比其他测试都多。因此，如果没有充足的时间实现整个回归测试的自动化，首先应该考虑实现烟雾测试的自动化；使用工具执行性能测试往往会容易很多；在某些环境中，如果不使用工具将不能执行负载测试。

2. 枯燥的任务

枯燥的任务也是自动化测试的主要候选对象，包括代码覆盖、数字计算、仿真和人力密集型任务等。如果不使用工具，这些任务将不可能在任何比较大的规模上进行。

5.1.3 自动化测试的优势和局限

通常，软件测试的工作量很大（据统计，测试会占用到 40％的开发时间，一些可靠性要求非常高的软件，甚至占到开发时间的 60％[16]），涉及面较广。同时，测试中有许多重复性的、非智力性的和非创造性的，并要求高准确细致的操作。由上节可知，这些任务最

适合于用计算机代替人工去完成，是自动化测试的主要候选对象。另外，自动化测试还能够完成大量手工无法完成的工作，如并发用户测试、大数据量测试、代码路径全覆盖测试、短时内大量测试、长时运行可靠性测试、与时序、死锁、资源冲突、多线程等有关的测试等。因此，自动化测试的引入是必然的，它具有以下一些优势[5,16]。

(1) 更方便回归测试。这可能是自动化测试最主要的任务，特别是在程序修改比较频繁时，效果是非常明显的。因为回归测试的动作和用例是完全设计好的，测试期望的结果也是完全可以预料的，所以将回归测试自动运行，可以极大提高测试效率，缩短回归测试时间。

(2) 提高测试质量。当软件被修改后，其变化经常不是很大，此时，自动化测试工具能以便利的方式验证新引入的错误，不仅节省了测试时间，而且使测试达到测试每个质量特性的目的，从而提高软件测试质量。

(3) 提高测试效率，缩短测试时间。随着软件系统的规模不断增大，功能不断增多，人工测试将非常耗时和烦琐，势必导致测试效率低下。在充分并合理地使用了测试工具后，不仅可以减轻手工测试工作，而且还可以把控制和管理引入整个测试过程，提高测试效率，加快测试的完成。

(4) 提高测试覆盖率。通过录制回放及数据驱动来测试功能，可以提高测试覆盖率。通过测试工具的辅助分析功能，可以提高测试的深度。

(5) 易于执行手工测试困难或不能完成的测试任务。有些非功能性方面的测试(如压力测试、负载测试、大数据量测试、崩溃性测试、大量用户的测试等)是人工测试不可能实现的。例如，对于大量用户的测试，很难甚至不能找若干台电脑和同样数目的操作人员在同一时刻进行测试，这样的测试方法不切实际且无法捕捉程序内部变化情况，但却可以通过自动化测试模拟同时有许多用户，从而达到测试的目的。

(6) 更好地利用资源。理想的自动化测试能够按计划完全自动地运行，自动化测试完全可以在周末或者晚上等时间段执行测试。将烦琐的任务自动化，可以提高准确性和测试人员的积极性，将测试技术人员解脱出来投入更多精力设计更好的测试用例。

(7) 更好地重现软件缺陷。由于每次自动化测试运行的脚本是相同的，所以每次执行测试的结果和内容具有一致性。由于自动化测试的一致性，很容易发现被测软件的任何改变。

(8) 提高软件测试的准确度和精确度，增加软件信任度。因为测试是自动执行的，所以不存在执行过程中的疏忽和错误，完全取决于测试的设计质量。一旦软件通过了强有力的自动测试后，软件的信任度自然会增加。

(9) 增进测试人员与开发人员之间的合作伙伴关系。为了更好地使用自动化测试工具，测试工程师需要对开发技术有深入的理解和实践，从而有了与开发工程师更多、更平等地交流的机会。自动化测试为测试工程师与开发工程师协同工作提供了一种交流和联系的手段，双方将有更多的合作与尊重。

(10) 运行更多更烦琐的测试。自动化的一个明显好处是可在较短时间内运行更多测试。

虽然自动化测试可以提高测试效率，能够完成手工测试困难或无法胜任的工作，但

它不是万能的，不能完全代替手工测试。在实际应用中，自动化测试也存在一定的局限性[5,16]。

（1）不能取代手工测试。

下列情况不适合于自动化测试。

① 测试很少进行。

② 测试结果适合于人工验证。

③ 软件不稳定。如果软件不稳定，则会由于这些不稳定因素导致自动化测试失败，或者致使测试结果本身就是无效的。

④ 涉及物理交互的测试。自动化测试工具不能很好地完成与物理设备的交互，如刷卡机的测试、断开设备物理连接的测试、开关电源的测试等。

⑤ 人体感观与易用性测试。界面的美观、声音的体验、易用性的测试，无法用测试工具来实现。

⑥ 定制型项目。为客户定制的项目，甚至采用的开发语言、运行环境也是客户特别要求的，开发公司在这方面的测试积累少。

⑦ 周期短的项目。项目周期短，测试周期将也很短，因此花大量精力准备的测试脚本，不能得到重复地利用。但为了特定测试目的而执行的测试任务除外，如针对特定应用的性能测试等。

⑧ 业务规则复杂的对象。业务规则复杂的对象有着较复杂的逻辑和运算关系，要么工具很难实现这些测试过程；要么需要投入比直接进行手工测试更长的测试准备时间。

（2）发现故障的数量不及手工测试。自动化测试主要是执行重复测试，因此被测软件在自动化测试中暴露的故障要少得多。自动化测试主要用于回归测试，进行正确性验证，而不是发现故障。据统计，自动测试只能发现约15%的故障，而手工测试可以发现85%的故障。

（3）不能提高测试的有效性。自动化测试只是用于提高测试的效率，即减少测试的开销和时间。

（4）不具有想象力，不能主动发现缺陷。自动化测试是通过测试工具进行，测试过程只是按照运行机制执行。自动化测试在许多情况下不能直接判断测试结果的正确性，还需要人工干预判断。自动化测试通常不能及时处理意外事件（如网络连接中断等），而且一般会被其中止。

（5）可能会制约软件开发。自动化测试的维护经常受到限制，制约软件的开发。

虽然自动化测试存在着上述种种局限，但是只要测试人员能够不断地提高测试技巧、测试理念以及被测系统的业务背景，合理配合使用手工测试和自动化测试，充分发挥各自的优势，那么就可以极大地减轻工作量，提高测试的质量、效率和可靠性，达到预期目的。

5.1.4 国内软件自动化测试实施现状分析

在国内，由于自动化测试发展时间不长，测试人员技能水平不高，软件自动化测试的研究和实施基本处于起步阶段。虽然也已出现了一批具有优秀的自动化测试实施经验的企业，然而还有很多企业和组织对自动化测试认识不足，甚至不愿接受。国内软件自

动化测试可分为三个层次[17]。

第一层次：测试的自动化。

其目的是用自动化工作代替原先手工工作。其显著的特征是以工具为中心。以QTP的应用为例，用QTP来完成原先人工操作执行的测试案例（如点击、录入等），若QTP不支持，就寻找解决方案，或更换其他测试工具。

另外，自动化测试不能完全彻底地代替手工测试，而必须要和手工测试整合在一起，才能反映出其价值。即自动化测试要全方位和手工测试执行，包括案例管理、测试执行以及报告呈现。

第二层次：自动化的测试。

其目的是将测试所涉及的各个环节作为一个统一的整体进行考虑，从案例管理，测试执行到报告呈现都有相应的策略及自动化的实现。这些策略思想、规范和代码的集合形成了一套自动化测试框架。

第三层次：软件流程框架。

整合软件自动化测试和软件开发，从自动化流程上，达到了真正的测试驱动开发级别，如单元测试（unit testing）与编码（coding）的整合。

当前，大多数国内软件企业的自动化测试实施处于第一层次或第一层次向第二层次的过渡，而国外微软、IBM等则已达到第三层次。国内软件企业实施或有意向实施自动化测试时遇到的问题主要有：

（1）因人员、资金、资源等的不足，认为不必实施，从而放弃实施。

（2）因一时热情，购买了自动化测试工具，实行了新的测试流程。然而，一段时间后，工具变成了共享资源，测试流程又回到旧模式。

（3）因开发与测试之间，甚至与项目经理之间矛盾重重，致使自动化测试实施的成本比手工测试更高，工作量更大，没有发挥出自动化测试的优势。

（4）实施较为成功，但依然存在一些问题。自动化测试流程得到了推行，但因工具不适合，文档不完备，培训和技术支持不充分，人员配备不合理，脚本可维护度不高等因素，影响了测试质量和测试效率的进一步提高。

5.1.5 软件自动化测试的引入条件

在引入并成功实现测试自动化时，必须综合考虑多方面的因素，尽可能地减少引入风险并可持续性地开展下去。杜文洁等[5,16]列出了以下一些需要考虑的因素。

1. 正确认识软件测试自动化

（1）自动化测试能极大地降低手工测试的工作，但绝不能完全代替手工测试。完全纯粹的自动化测试仅仅是理论上的目标，实际操作上不仅代价昂贵，而且基本不可能。一般来说，利用自动化的程度超过60%将过大的增加测试相关的维护成本。

（2）自动化的引入有一定的标准。要经过综合的评估，绝对不能理解成测试工具简单的录制与回放过程。实际上，从实现成熟度来说，自动化测试分五个级别[18]，如表5-1所示。

表 5-1　自动化测试级别表

级别	说　　明	优　　点	缺　　点
一级	录制和回放	自动生成自动化测试脚本，而不需要任何编程知识	测试脚本较大，当需求和应用发生变化时，相应的测试脚本也必须被重新录制
二级	录制、编辑和回放	减少脚本的数量和维护的工作	需要一定的编程知识；频繁的变化难于维护
三级	编程和回放	确定了测试脚本的设计，在项目的早期就可以开始自动化测试	要求测试人员具有很好的软件技能，包括设计、开发等
四级	数据驱动	能够维护和使用良好的并且有效模拟真实生活中数据的测试数据	软件开发的技能是基础并且需要访问相关的测试数据
五级	动作词的测试自动化	测试用例的设计与测试工具被分离开	需要一个具有工具技能和开发技能的测试团队

(3) 自动化测试能快速定位被测软件的功能与性能缺陷，但不会创造性地发现脚本没有设计的缺陷。测试工具要求设计者定制各种分支路径的校验点，如果没有定制完整，即使存在明显的缺陷，也不会发现。因此，测试前必须制定全面、系统的测试设计。

(4) 自动化测试不适于周期短、时间紧迫的项目。与自动化测试相关的工作量很大，将企业级自动化测试框架应用到任何一个项目中也要评估其合适性，因此不能盲目地应用到任何一个测试项目中，尤其不适合周期短的项目。

(5) 必须进行多方面的培训，包括测试工具使用、测试流程、缺陷管理、人员安排等。

2. 评估分析企业自身的现状

(1) 企业规模不限。不管公司大小，关键是从 CTO 到普通工程师，一定要有实施自动化测试的坚定决心。一般来说，总人数不少于 10 人，测试人员与开发人员比例合适(比如 1∶1 到 1∶1.5)的软件开发团队可以优先开展自动化测试工作。

(2) 开发产品的企业比开发项目的企业优先实施自动化。主要原因在于测试维护成本和风险小。因为产品软件开发周期长，需求相对稳定，测试人员有较充裕的时间去设计测试方案和开发测试脚本；而项目软件面向客户，需求变更频繁，一般以开发代码为主，很难照顾到测试代码。只要开发流程、测试流程、缺陷管理流程规范，做项目软件的企业推行自动化测试也相对轻松。当前国内做项目的软件公司居多，有很多正在推行 CMM 等级标准。

(3) 开发和管理流程标准化。不管是 CMM 还是 ISO，不管是开发流程、测试流程还是缺陷管理流程，都可以参考 RUP(Rational Unified Process，Rational 统一过程)以及很多业界文献。一个很好的工作原则是把想做的写下来(计划管理)，按照写下来的去做(行为管理)，把做的事情记录下来(报告管理)，出现的问题要设法解决(跟踪管理)。如果软件开发团队据此开发软件，那么就完全具备实施自动化测试的条件。

3. 评估软件自动化测试切入方式的风险

自动化测试与手工测试需要配合使用。对于自动化测试率的目标是 10/90(10% 的

自动化测试和90% 的手工测试)。当这些目标都实现了,可以将自动化测试的使用率提高。

一般来说,在下列测试条件下,可以考虑引入自动化测试。

(1) 非常重要的测试;

(2) 项目没有严格的时间压力;

(3) 具有良好定义测试策略和测试计划的测试;

(4) 拥有一个能被识别的测试框架和候选者;

(5) 能够确保多个测试运行的构建策略;

(6) 运行频繁的测试;

(7) 多平台环境需要被测试;

(8) 拥有运行测试的硬件;

(9) 拥有关注在自动化过程上的资源;

(10) 能自动化或易于自动化的测试。

反之,如果以上部分条件不满足甚至全部都不满足,则宜采用手工测试。

4. 评估企业软件开发语言的风险

当前业界流行的测试工具有几十种,而相同功能的测试工具所支持的环境和语言各不相同。当前国际上流行的几个软件测试工具生产厂商及一些主要IDE产品信息[18],如表5-2所示。

表5-2 软件测试工具生产厂商及其主要IDE产品信息表

生产厂商	工具名称	测试功能简述
Mercury Interactive Corporation	WinRunner	功能测试
	LoadRunner	性能测试
	QuickTest Pro	功能测试
	Astra LoadTest	性能测试
	TestDirector	测试管理
IBM Rational	Rational Root	功能测试、性能测试
	Rational XDE Tester	功能测试
	Rational TestManager	测试管理
	Rational PurifyPlus	白盒测试
Compuware Corporation	QARun	功能测试
	QALoad	性能测试
	QADirector	测试管理
	DevPartner Studio Professional	白盒测试

续表

生产厂商	工具名称	测试功能简述
Seque Software	SilkTest	功能测试
	Silk	性能测试
	SilkCentral Test Issue Manager	测试管理
Empirix	e-Tester	功能测试
	e-Load	性能测试
	e-Monitor	测试管理
Parasoft	Jtest	Java 白盒测试
	C++ test	C/C++ 白盒测试
	.NETtest	.NET 白盒测试
RadView	WebLoad	性能测试
	WebFT	性能测试
MicroSoft	WebApplication Stress Tool	性能测试
Quest Software	Benchmark Factory	性能测试
Minq Software	Pure	功能测试、性能测试、测试监控
Seapine Software	QA Wizard	功能测试
	TestTrack Pro	缺陷管理

5. 估算时间

估算自动化测试实施的时间周期，防止时间的浪费。

6. 评估工作流程变更风险

实施自动化测试时，测试工具的工作流程经常会引起团队的测试流程、开发流程的相应变更。而这种变更可能会引起团队成员的抱怨情绪。因此，应尽量减少这种变更，并解决好变更中可能存在的困难。

7. 人员培训与变更风险

提前对测试团队人员培训和实施过程中人员变更的风险性做出预测和相应的处理方案。

一个企业或软件团队实施自动化测试，必然会面临重重压力和风险，但是只要团队成员和公司高层坚定自动化测试的决心和信心，事先做好评估和风险预测，就一定能成功地实施自动化测试，提高测试的效率和质量。

5.1.6 自动化测试的运用步骤

自动化测试的运用步骤[5]包括：

1. 完善软件测试过程

开始自动化测试前，需要完善测试计划和过程，明确测试设计和预期结果的描述，指明测试所需的数据，给出设计数据的完整方法，完成测试设计文档，并确定所采用的测试方法。其中，测试设计是最主要的测试需求说明。测试执行过程的步骤说明可以简化，只要有软件基本操作常识的人员能根据文档完成执行工作即可。

采用较多的计算机是一个提高测试效率的简单方法。自动化测试需要集中考虑所需要的支撑设备。测试设备、测试环境选择不当可能会直接导致测试结果无效甚至错误。例如，企图使用单机完成一些大容量的自动化测试执行工作。

改进被测产品的性能。当被测产品的性能对测试速度的影响达到一定程度时，可将其作为影响测试进度的缺陷，提交缺陷报告。

改进被测产品的功能。当被测软件产品的功能较难实现或者需要测试人员花费较多时间去实现时，与其实现该功能的自动化测试，不如改进该功能。

2. 定义需求

实际测试中，应该有一份自动化测试需求。它应在测试设计阶段被详细定义出来，用以描述自动化测试的目标。

自动化测试前，应该明确测试成功的标准。

手工测试和自动化测试的配合使用是最好的方法。要么自动化执行，手工确认执行结果；要么手工执行，自动化确认测试结果。手工测试往往能够发现一些不引人注意的问题。

3. 验证概念

自动化测试前，必须验证其可行性，包括验证所采用的测试工具和方法的可行性，验证被测产品适合自动化测试的可行性。验证过程往往需要较多的帮助，花费时间较长。用来验证概念的测试方案是评估测试工具的最好的方式。

下面是一些候选的验证概念的试验：

(1) 回归测试。它是自动化测试的最主要任务。

(2) 配置测试。所有的测试用例都应该在其支持平台上被执行。

(3) 搭建测试环境。当大量的测试用例所需要的测试环境相同时，先实现环境搭建的自动化是个较好的选择。

(4) 非 GUI 测试。非 GUI(CLI 和 API)测试自动化的实现比 GUI 要容易得多。

4. 支持产品的可测试性

软件产品一般使用命令行接口(Command Line Interface，CLI)、应用程序接口

(API)和图形用户接口(GUI)三种不同类别的接口。与 GUI 相比,CLI 和 API 容易实现自动化。因此,应该鼓励开发人员在产品中提供 CLI 和 API,以支持产品的可测试性。有时它们被隐藏在产品的内部。

5. 设计的可延续性

为了与产品新版本的功能和其他相关的修改保持一致,自动化测试需要被长期不停地维护和扩充。

6. 有计划地部署

有计划地部署,确保产品相关人员能够获得测试方案。

7. 开展自动化测试

通过合理地引入测试工具,顺利地完成自动化测试,能够提高测试质量,缩短测试时间,从而更快、更好地为用户提供高质量的软件产品。

近年来,国内软件测试技术发展迅速,并从纯手工测试向测试自动化、工程化方向发展。要真正实现软件测试的自动化、工程化,就必须要有一批优质的软件测试自动化、工程化的工具。下面就对软件测试工具的情况进行介绍。

5.2 软件自动化测试工具

软件自动化测试工具是实现软件自动化测试的关键因素。因此,选择一款优质的、适合测试项目实际情况的测试工具是成功实现自动化测试的第一步。本节将对自动化测试工具的作用、分类及常用的自动化测试工具进行介绍。

5.2.1 自动化测试工具的作用及优势

一般而言,软件自动化测试是借助测试工具进行的。部分的测试设计、实现、执行和比较都能够用测试工具执行。人工设计的测试用例由工具执行并对结果进行比较。如果采用自动比较技术,可以自动完成对测试用例执行结果的判断,避免人工比较存在的疏漏问题[16]。

自动化测试工具的作用主要体现在以下几个方面:

(1) 确定最优硬件配置。

(2) 检查可靠性。

(3) 检查软硬件的升级情况。

(4) 评估新产品。

自动化测试工具的优势主要有:

(1) 记录业务流程并自动生成脚本程序。

(2) 精确测试。

(3) 生成大量虚拟用户。
(4) 监控软硬件系统。
(5) 模仿网络设备。
(6) 胜任人工无法完成的困难测试。
(7) 显示和分析测试结果。

5.2.2 自动化测试工具分类

目前,自动化测试工具的种类繁多。按测试方法可分为白盒测试工具和黑盒测试工具[16];按收费方式可分为商业测试工具、开源测试工具和免费测试工具;按测试的对象和目的可分为单元测试工具、负载测试工具、功能测试工具、性能测试工具、回归测试工具、Web 测试工具、数据库测试工具、嵌入式测试工具、测试设计与开发工具、测试执行和评估工具、测试管理工具等。

1. 白盒测试工具

白盒测试工具一般应用于被测源程序,针对程序代码、程序结构、对象属性等,测试所发现的缺陷可以定位到代码行、对象或变量级。根据工作原理的不同,又可分为静态测试工具和动态测试工具。

1) 静态测试工具

静态测试工具是在不运行被测程序的前提下,通过检查程序代码,分析软件的特性,找出缺陷,得出测试报告。静态分析主要集中在需求文档、设计文档、程序结构以及代码语法等方面。按照完成的职能不同,静态测试工具包括以下几种类型:

(1) 代码审查;
(2) 一致性检查;
(3) 错误检查;
(4) 接口分析;
(5) 输入输出规格说明分析检查;
(6) 数据流分析;
(7) 控制流分析;
(8) 类型分析;
(9) 单元分析;
(10) 复杂性分析;
(11) 规范性分析。

2) 动态测试工具

动态测试工具通过实际运行被测系统,并设置断点,向代码生成的可执行文件中插入一些监测代码,掌握断点时刻程序运行的数据(对象属性、变量的值等),具有功能确认、接口测试、覆盖率分析、性能分析等性能。动态测试工具可以分为以下几种类型:

(1) 功能确认与接口测试;
(2) 覆盖测试;

（3）性能测试；
（4）内存分析。

2．黑盒测试工具

黑盒测试工具是在明确被测软件产品应有功能的前提下，完全不考虑其内部结构、内部特性和处理过程，来检验软件功能是否按需求说明正常工作。

按照完成的职能不同，黑盒测试工具可分为功能测试工具和性能测试工具。功能测试工具用于测试程序的功能是否达到要求；性能测试工具用于测试软件和系统的性能如何。

3．商业测试工具

需要付费，且价格较昂贵，但相对成熟和稳定，且提供售后服务和技术支持。

目前，商业测试工具在GUI功能测试和性能测试方面占有较多。流行的基于GUI的功能自动化测试工具有Robot、QTP、TestComplete等。各种自动化测试工具在IDE、脚本开发语言、脚本开发方式、支持的控件等方面有较大区别，但其实现的功能大致相同。

4．开源测试工具

源代码公开发布，由自愿者开发和维护的软件。它是测试工具的一个重要分支。越来越多的软件企业开始使用开源测试工具。只要不作为商业用途，大多数开源测试工具可以免费使用，有时需要考虑使用的成本，甚至某些方面可能比商业测试工具的成本还要高。

相对于商业测试工具而言，开源测试工具有几个明显的优势：
（1）成本更低。在不作商业用途的前提下，大部分开源测试工具可免费使用。
（2）选择余地更大。能打破商业测试工具的垄断地位，给测试人员更多的选择空间。
（3）可改造性好。源代码开放，可按需求对其进行个性化改造。
其不足之处在于：
（1）安装和部署相对困难。大部分开源测试工具的安装配置等过程都较复杂。
（2）易用性不足。易用性、用户体验方面偏差。
（3）稳定性不强。部分开源测试工具不够稳定。
（4）学习和获取技术支持较难。大部分开源测试工具缺乏培训指导和技术支持服务，仅带有粗糙的联机帮助和用户手册，给测试人员的学习增加了一定的难度。

5.2.3 常用自动化测试工具简介

本节将根据各测试工具的供应商网站[19-22]及其使用手册[23,33]提供的信息，对常用的自动化测试工具的特点、功能等进行简单介绍。

1．Rational Robot

Rational Robot是IBM/Rational的产品之一，是QA团队进行C/S应用程序功能测

试的自动化测试工具。支持缺陷检测，包括测试用例和测试管理，支持多项 UI 技术。

Rational Robot 可用于功能测试的 GUI 脚本、性能测试的 VU 和 VB 脚本进行开发。其特性和作用有：

（1）缩短测试人员发现测试自动化流程的价值的学习进程。

（2）支持测试自动化工程师通过扩展测试脚本检测缺陷和定义测试用例。

（3）提供针对通用对象的测试用例和用于开发环境对象的专门测试用例。

（4）包含内置测试管理，集成了 IBM Rational Unified Process 工具。

（5）协助进行缺陷跟踪、变更管理和需求跟踪。

（6）简化测试配置。可用于多台机器间按照不同配置来执行功能测试。相同的功能测试可以同时执行以缩短指定配置验证问题的时间。

（7）支持 HTML、DHTML、VB、Java、VS. NET、VC++、Oracle Developer、PowerBuilder、Delphi 等广泛的环境和语言开发的应用程序以及用户界面上看不见的那些对象。

（8）支持常见控件和对象的测试。可用于变化条件下的应用程序组件的测试，并提供菜单、列表、文本字符、位图等多种对象的测试用例。

（9）提供集成的编程环境。当录制时，允许以一种方便浏览和编辑测试脚本的集成 MDI 编程环境 SQABasic 生成测试脚本。

（10）有助于快速分析问题。自动日志及测试报告和脚本代码颜色区分有助于快速预览和分析问题。通过双击一个实体，可以直接定位到测试脚本，有助于快速分析和更改测试脚本错误。

2. WinRunner

WinRunner 是 Mercury Interactive 公司提供的一种检验应用程序是否按预期运行的企业级功能测试和回归测试工具。通过自动录制，检测和回放用户的操作，它能有效地识别缺陷并且确保那些跨越多个应用程序和数据库的业务流程在初次发布就能避免出现故障，以及保持长期可靠运行。

2006 年 Mercury 公司被 HP 全权收购。目前 WinRunner 已从 HP 产品家族中消失，然而，国内外仍有众多公司使用它进行自动化测试。它的 C 语言脚本为 IT 系统底层及嵌入式领域的应用提供了强大的便利性。

WinRunner 能快速、批量地完成功能点测试；能按照同一脚本重复执行相同的动作，消除人工测试带来的误差；能自动重复执行工作任务，减少测试时间；能用简单的测试工具覆盖不同的环境优化测试；通过修改和重用测试脚本，使测试投资最小化；支持程序风格的测试脚本，通过使用通配符、宏、条件语句、循环语句等，能较好地完成测试脚本的重用。其主要功能有：

1）轻松创建测试

通过点击鼠标和使用键盘完成一个标准的业务操作流程，就能创建测试。WinRunner 能自动记录该操作并生成脚本，并能直接修改脚本以满足各种复杂测试的需求。针对测试团队中业务用户和专业技术人员的不同需求，WinRunner 提供了两种不同

的测试创建方式。

2）插入检查点

测试录制时，通过插入检查点，可检查应用程序是否正常运行。WinRunner 提供了文本、GUI、位图和数据库等类型检查点。测试运行时，WinRunner 会自动收集数据指标对检查点进行一一验证。

3）检验数据

WinRunner 能验证数据库的数值，以确保业务交易的准确性。

4）增强测试

WinRunner 的数据驱动向导(Data Driver Wizard)可以把一个业务流程测试转化为数据驱动测试，从而反映多个用户各自独特且真实的行为。数据驱动测试不仅节省了时间和资源，又提高了应用的测试覆盖率。

以系统登录流程为例，将登录名和密码作为可变栏，用多套数据进行测试。使用 Data Driver Wizard，选择用数据表格文件中与登录名和密码对应栏目的数据替换，将登录名和密码输入数据表格文件，或从其他表格和数据库中导入。

WinRunner 提供了 Function Generator 以增加测试的功能，提供了 Virtual Object Wizard 来识别未知的非标准对象。

5）运行测试

WinRunner 会根据测试运行时的业务流程自动操作应用程序。

6）分析结果

通过交互式的报告工具，WinRunner 提供了详尽、易读的报告。报告中会列出测试中发现的错误内容、位置、检查点和其他重要事件，帮助对测试结果进行分析。

7）维护测试

应用程序被修改后，需要再次被测试。WinRunner 能创建可重复使用的测试，而无须在每次改动程序后重新创建，从而极大地节省了时间和资源，充分利用测试投资。

另外，WinRunner 将应用对象保存为 GUI Map 文件以记录测试。通过修改 GUI Map 文件，WinRunner 可以方便地实现测试重用。此外，WinRunner 还能为基于无线的应用程序提供帮助。

3. LoadRunner

至 2011 年 6 月，其最新版本为 HP LoadRunner 11.00。HP LoadRunner 可以在新系统或升级部署之前找出瓶颈所在，从而防止在生产过程中出现代价高昂的应用程序性能问题；能够测量端对端性能、诊断出应用程序及系统瓶颈并让其发挥更好的性能。集成的载入测试、性能测试和应用程序应力测试功能将有助于减少在生产环境中测试和部署新应用程序和系统所需的成本和时间。IBM、Sun、Oracle 等著名公司都使用该软件，但其价格昂贵。业界认为，其功能与 QALoad 相比不相上下。

其主要功能有：

1）轻松创建虚拟用户

Virtual User Generator 允许简便地创建系统负载，能生成虚拟用户，并用虚拟用户

模拟真实用户的业务操作行为。首先，记录业务流程（如下订单等）；然后，将其转化为测试脚本。利用虚拟用户，可在 Windows、UNIX 或 Linux 上同时产生成千上万个访问用户。

可以对虚拟用户进行参数化操作，以用不同的实际发生数据来测试应用程序，从而反映出系统的负载能力。以系统登录为例，参数化操作可将记录中的登录名和密码等固定数据由变量来代替。在这些变量内随意输入可能的登录名和密码，来匹配多个实际用户的操作行为。

为进一步确定虚拟用户能够模拟真实用户，可利用 LoadRunner 控制某些行为特性。例如，只需要点击鼠标，就能轻易控制交易数量、交易频率、用户的思考时间和连接速度等。

2）模拟生产工作负载

LoadRunner 可以设定负载方案、业务流程组合和虚拟用户数量。能快速组织起多用户的测试方案，既能建立起持续且循环的负载，又能管理和驱动负载测试方案。可定义用户在何时访问系统以产生负载，实现测试过程自动化。能监测系统架构中各个组件的性能，包括服务器、数据库、网络设备等。

3）识别性能瓶颈

内置集成的实时监测器，允许随时观察应用系统的运行性能，实时显示交易性能数据（如响应时间等）和其他组件的性能。在测试过程中，可以从客户和服务器两方面评估系统组件的运行性能，以便较快地发现问题。

4）诊断问题的根本原因

测试结束后，LoadRunner 汇总所有的测试数据，并提供高级的分析和报告工具，以便快速查找问题并诊断其根本原因。通过使用 LoadRunner 的分析工具，能快速地查找问题和原因并作出相应的调整。

5）部署前提高应用程序性能

负载测试是一个重复的过程。每次改正错误后，都必须在相同的方案下，对应用程序再次执行负载测试，以检验所做的修正是否改善了运行性能。

LoadRunner 完全支持 EJB（Enterprise Java Beans）的负载测试。允许重复执行错误改正前后相同的测试方案，并通过 HTML 报告提供性能结果比较基准。支持无线应用协议 WAP 和 I-mode，支持 Media Stream，可以记录和重放任何流行的多媒体数据流格式来诊断系统的性能问题，诊断原因、分析数据的质量。

4. QACenter

QACenter 是 Compuware 公司提供的黑盒测试工具。它能创建快速、可重用的测试过程，能管理测试过程，快速分析和调试程序，能针对回归测试、强度测试、单元测试、并发测试、集成测试、移植测试容量和负载测试建立测试用例，自动执行测试并产生相应的测试文档。

QACenter 测试工具主要包括以下几个模块。

1）功能测试工具 QARun

QARun 主要用于客户端/服务器（Client/Server，C/S）系统中客户端的功能测试，包

括测试系统的 GUI 及事务逻辑。通过鼠标移动、键盘操作被测系统,得到相应的脚本,并可对脚本进行编辑和调试。插入检查点时,可建立期望输出值(基线)。执行检查点后,可确定实际结果与期望结果是否相同。

2）性能测试工具 QALoad

主要用于主从应用系统、企业资源规划(ERP)和电子商务应用软件的自动化负载与压力测试。通过模拟大量用户执行关键业务,对应用程序进行负载测试。通过反覆的模拟测试,严格测试应用系统的可扩展性和系统性能。其主要功能和特性有:

(1) 操作简便

利用 QALoad Script Development Workbench 创建完整的功能脚本。捕获会话并转换到脚本,修改和编译脚本完成测试脚本开发,将编译好的脚本分配至测试环境的机器上,驱动多个 play agent 模拟大量用户的并发操作,实施应用的负载测试。由此可见,其操作简便,减轻了大量的人工工作,节省了时间,提高了效率。

(2) 适用性广

广泛支持 DB2、DCOM、ODBC、Oracle、NETLoad、Corba、QARun、SAP、SQL Server、Sybase、Telnet、TUXEDO、UNIFACE、WinSock、WWW 等。

(3) 预测系统性能

当应用升级或者新应用部署时,负载测试帮助确定系统是否按计划处理用户负载。QALoad 能够模拟数以千计的用户进行商业交易。

(4) 通过重复测试寻找瓶颈问题

QALoad 的录制和回放能力提供了验证负载下应用性能的可重复方法,能模拟大量用户,并执行和运行测试。通过反复测试可以充分地测试与容量相关的问题,快速确认性能瓶颈并进行优化和调整。

(5) 从控制中心管理全局负载测试

QALoad Conductor 工具能定义、管理和执行负载测试;能自动识别网络中可进行负载测试的机器,并自动分布工作量,以避免网段超载;能自动启动和配置远程用户,进行全球负载测试,并收集有关性能和时间的统计数据。

(6) 验证应用的可扩展性

为了提高应用的可扩展性,QALoad 提供了远程存储虚拟用户响应时间并在测试结束或特定时间下载这些资料的功能。

另外,QALoad 还有集成的系统资源视图,能够快速产生虚拟负载测试。

3）测试的组织设计和创建及管理工具 QADirector

QADirector 提供应用系统管理框架,能组合所有测试阶段,从而最有效地使用现有测试资料、测试方法和应用测试工具。使用 QADirector 能自动地组织测试资料,建立测试过程;能按正确的次序执行多个测试脚本,记录、跟踪、分析和记录测试结果,并与多个并发用户共享测试信息。

4）可用性管理工具 EcoTools

系统的可用性分析一般在性能测试后进行。系统的可用性受用户桌面、网络、服务器、数据库环境以及各种子组件等因素的影响。

EcoTools 是 EcoSystem 组件产品的基础，用于解决应用可用性中的计划、管理、监控和报告的问题，但不用于诊断问题。其与 EcoTools 集成能为加载测试和计划项目需求提供全面的解决方案。

EcoTools 可以监控服务器资源，包括监控 Windows NT、UNIX、Oracle、Sybase、SQL Server 和其他应用包。

5）性能优化工具 EcoScope

EcoScope 是一套定位于应用及其依赖的所有网络计算资源的解决方案，可以提供应用视图，并标出应用是如何与基础架构相关联的，能解决复杂环境下分析与测量应用系统性能的难题。

EcoScope 能无干扰地监控网络，自动跟踪 LAN/WAN 的应用流量，采集详细的性能指标，并将这些信息关联到交互界面中，自动识别低性能的应用系统、受影响的服务器及用户性能低下的程度。另外，用户界面能智能地访问大量的 EcoScope 数据。因此，EcoScope 能较快地找到性能问题的根源，其应用主要表现在确保成功部署新应用、维护性能的服务水平、加速问题检测与纠正和高效地分析数据等方面。

6）测试数据生成工具 TestBytes

通过点击操作可确定数据类型(包括特殊字符的定制)，通过与数据库的连接可自动生成数百万行正确的测试数据。TestBytes 能极大地提高数据库开发人员、QA 测试人员和应用开发人员的工作效率。

5. QuickTest Professional

QuickTest Professional 是 HP Mercury 提供的一款用于创建功能和回归测试的自动化测试工具。它自动捕获、验证和重放用户的交互行为。

目前，QuickTest Professional 最新的版本为 11.0。QTP 是业内市场份额最大的测试工具，是目前 HP 功能测试软件的绝对主力。支持功能测试和回归测试自动化，用于每个主要软件应用程序和环境。此解决方案使用关键字驱动的测试概念，简化了测试创建和维护过程。它使测试人员能够使用专业的捕获技术直接从应用程序屏幕中捕获流程来构建测试案例。测试专家还可通过集成的脚本和调试环境完全访问内在测试和对象属性。

QTP8.0 提出了自动化测试领域革命性的一个新名词——关键字驱动。

QTP 进行功能测试的测试流程大致分制定测试计划、创建测试脚本、增强测试脚本、运行测试和分析测试结果五个步骤[24]。

1）制定测试计划

根据被测项目的具体需求和所用测试工具制定测试计划，用于指导测试全过程。

QTP 不能完全取代测试人员的手工操作，但是在某个功能点上，QTP 能帮助测试人员完成很多重复和冗余的工作。分析被测程序的特点，确定需要 QTP 进行测试的那些功能点，可以细化到具体页面或控件。就普通的应用程序而言，QTP 应用在某些界面变化不大的回归测试中是非常有效的。

2）创建测试脚本

当测试人员浏览站点或操作应用程序的时候，QTP的自动录制机制能将其每一步操作及被操作的对象记录下来，并自动生成测试脚本。与其他自动测试工具录制脚本不同的是，QTP除了以VBScript方式生成脚本外，还将被操作的对象及相应的动作按照层次和顺序保存在一个基于表格的关键字视图中。例如，测试人员单击一个链接，然后选择一个CheckBox控件，这样的操作流程都会完整地被记录在关键字视图中。

3）增强测试脚本

录制完毕后，测试人员可以根据需要增加一些扩展功能。QTP允许测试人员通过在脚本中增加或更改测试步骤来修正或自定义测试流程。例如，通过增加多种类型的检查点，既可以检查在程序的某个特定位置或对话框中是否出现了所需的文字，还可以检查一个链接是否返回了正确的URL地址；通过参数化，使用多组不同的数据驱动整个测试过程。

4）运行测试

运行测试时，QTP将从脚本的第一行语句开始执行，验证设置的检查点，用实际数据代替参数值，并给出相应的输出结果信息。QTP提供脚本调试功能。

5）分析测试

运行结束后，QTP会自动生成一份详细的测试结果报告。

QTP可以使新测试人员在几分钟内提高效率。只需通过简单地按“录制”按钮，并使用执行典型业务流程的应用程序即可创建测试脚本。系统使用简明的英文语句和屏幕抓图来自动记录业务流程中的每个步骤。用户可以在关键字视图中轻松修改、删除或重新安排测试步骤。

QTP可以自动引入检查点，以验证应用程序的属性和功能，也可以为任何对象添加几种类型的检查点，以便验证组件是否按预期运行。对于关键字视图中的每个步骤，活动屏幕均准确显示测试中应用程序处理此步骤的方式。

QTP可以在不需要编程的情况下处理数据集和创建多个测试迭代，从而扩大测试案例范围；可以输入数据或从数据库、电子表格或文本文件中导入数据。

高级测试人员可以在专家视图中查看和编辑测试脚本，该视图显示QTP自动生成的基于业界标准的VB脚本。专家视图中进行的任何变动自动与关键字视图同步。

脚本运行结束，TestFusion报告显示高级结果概述、准确指出应用程序故障位置的可扩展树视图、使用的测试数据、突出显示任何差异的应用程序屏幕抓图以及每个通过和未通过检查点的详细说明。

QTP加快了更新流程。当测试中应用程序出现变动（如“登录”按钮重命名为“登入”）时，可以对共享对象库进行一次更新，然后此更新将传播到所有引用该对象的脚本。用户可以将测试脚本发布到Mercury TestDirector，使其他QA团队成员可以重复使用测试脚本，从而消除了重复工作。

QTP自身又带有数据表支持数据驱动的测试，数据驱动使得自动化测试代码复用率显著提高。

QTP支持所有常用环境的功能测试，包括Windows、Web、.NET、Visual Basic、

ActiveX、Java、SAP、Siebel、Oracle、PeopleSoft 和终端模拟器。

6. TestDirector

TestDirector 是 Mercury Interactive 公司提供的一种企业级测试管理工具,也是业界第一个基于 Web 的测试管理系统[8]。Mercury 被 HP 收购后,HP TestDirector 现已被 HP Quality Center 取代。

TestDirector 通过提供对收集需求、计划及安排测试、分析结果和管理缺陷及问题的可重复流程,允许快速高效地部署高质量的应用程序。

TestDirector 是针对测试管理必不可少的各个方面——需求管理、测试计划、测试实验室和缺陷管理的一款独立的基于 Web 的应用程序。

TestDirector 支持 IT 团队间高水平的交流和协作,有助于横跨地域和组织范围的信息交互。

1) 需求管理

需求驱动整个测试。TestDirector 的 Web 界面简化了需求管理过程。

2) 测试计划的制定及测试用例的设计

其 Test Plan Manager 指导测试人员如何将应用需求转换为测试计划,组织起明确的任务和责任,并在测试计划制定期间为测试小组提供关键点和 Web 界面以协调团队间的沟通。

3) 人工与自动测试的结合

TestDirector 提供自动化切换机制,能让测试人员决定哪些重复的人工测试可转变为自动脚本,并能简化该转变,立即启动测试设计过程,以提高测试速度。

4) 安排和执行测试

TestDirector 的测试实验室管理提供了一个基于 Web 的安排测试日程的框架。其 Smart Scheduler 能根据测试计划指标对运行着的测试执行监控,能自动分辨是系统错误还是应用错误,然后将测试切换到网络的其他机器。

5) 缺陷及问题管理

TestDirector 的缺陷管理贯穿测试发现问题、修改错误和验证修改结果的全过程。测试人员只需进入一个 URL,就能够汇报和更新错误,过滤整理错误列表并作趋势分析。

6) 图形化和报表输出

TestDirector 的图表和报告有助于分析数据信息,并以标准的 HTML 或 Word 形式生成和发送正式测试报告。分析数据能简便地被输入到 Excel、ReportSmith、CrystalReports 和其他第三方标准化的工具中。

7. Jtest

Jtest 是 Parasoft 公司推出的一套用于提高开发团队效率和软件质量的经大量实践验证的自动化集成解决方案,重点在确认 Java 代码和应用程序,可以和 Parasoft SOAtest 无缝集成完成复杂分布式应用程序和事务的功能测试和负载测试。它是一款针对 Java 应用程序的自动化白盒测试工具。

其基本特性和功能有：

（1）静态分析。静态代码分析和数据流静态分析。

（2）单元测试。创建、执行、优化和维护 JUnit 测试。

（3）实时错误检测。检测并阻止影响应用程序安全性、可靠性和性能的缺陷，包括异常、资源及内存泄漏、安全弱点、性能等。

（4）支持大型团队开发中测试设置和测试文件的共享。

（5）内置支持 Google Android、Spring、Hibernate、Eclipse、TDD、JSF、Struts、JDBC、EJB 和 JSP 等。

（6）针对 Java 检查，内置静态分析规则配置集。

（7）提供 OWASP、CWE-SANS、PCI DSS 和其他安全静态标准的模板。

（8）自动纠正违反 QuickFix 规则的错误纠结种类超过 350 个。

（9）允许通过 GUI 方式或自定义方式来定制编码规范。

（10）提供 GUI 接口和命令行模式。

（11）可生成 HTML、PDF 和 XML 报告。

（12）团队或组织内共享测试设置和测试文件。

（13）自动执行回归测试。

（14）可与 IBM Websphere Studio 和 Eclipse IDE 集成。

第6章



功能测试

功能测试(functional testing)是指通过对程序进行功能抽象,将其划分为功能单元,然后通过数据抽象,对每个功能单元生成测试数据(用例)进行测试,检查程序产品是否达到用户要求的功能。测试时,将测试对象视为打不开的黑盒,只需测试产品的功能,而不需要了解其内部结构和处理过程,因此又称为黑盒测试。

功能测试的目的主要用于检查实际软件的功能是否符合用户的需求。如果曾以手动方式测试过应用程序或网站,就会很清楚手工测试方式的缺点——既耗时又乏味,而且需要投入大量的人力资源。最为糟糕的是,因时间的限制,在应用程序发布前以手工方式往往无法彻底地测试其所有功能。

现有软件功能测试工具较多,但基本原理都是以录制和回放的方式实现自动化功能测试。Mercury 的 QuickTest Professional(QTP)就是一款功能测试工具。使用 QTP 进行自动测试,可以极大地加快测试流程,从而解决这些问题。可以创建用于检查应用程序或网站所有方面的测试,然后在每次程序或网站更改时运行这些测试即可。运行测试时,QTP 将模拟实际用户的操作(如在应用程序或网站窗口中移动鼠标光标、单击图形用户界面对象和键盘输入),且比任何实际用户操作都快[23]。

本章将详细介绍功能测试工具 QTP。

6.1 QTP 简介

QTP 是 Mercury 公司开发的一款先进的自动化测试工具。目前该公司已被惠普(HP)收购。惠普官方网站对该工具做了一些介绍。

Mercury QuickTest Professional 企业级自动化测试工具针对功能测试和回归测试自动化提供业界最佳的解决方案,采用关键字驱动(keyword-driven)测试的概念,能完全简化测试的创建和维护工作,适于所有主要软件应用程序和环境的测试。

QTP 的关键字驱动测试方法可通过与关键字视图双向同步集成的脚本和调试环境,测试自动化专家对内在测试和对象属性具有完全访问权限。QTP 通过对 Web 页面或应用程序所进行的操作录制成自动化测试脚本,然后运行回放测试脚本,并可以在其中插入各种检查点来实现对 Web 页面或应用程序的功能的检查。

QTP 进行功能测试的测试流程主要可分为制定测试计划、创建测试脚本、增强测试

脚本、运行测试和分析测试结果五个步骤[24]。部分详细信息可参考第 5 章关于测试工具简介中的 QTP 介绍。

6.2 QTP 安装

本节将详细介绍 HP 网站提供的自动化功能测试工具 QuickTest Professional 9.2 试用版的安装。根据该软件安装的流程，整个安装过程可以划分为两个步骤。

1. 安装 Microsoft.NET Framework 2.0

在安装 QTP 测试工具之前，必须确保安装了 Microsoft.NET Framework 2.0 程序。运行图 6-1 中“QuickTest Professional 安装程序”，按照安装向导默认安装，直至出现“安装完成”提示。



图 6-1 QuickTest Professional 安装向导页面图

2. 安装 QTP

再次运行图 6-1 中的“QuickTest Professional 安装程序”，接受许可证协议后，在注册信息页填入维护号(序列号)：0123-2820602186，如图 6-2 所示。

按照安装向导默认安装，直至“客户注册”。无须注册，完成安装。重启计算机，QTP 会自动完成剩下的配置。至此，试用期 14 天的 QTP 安装完成。

图 6-2 维护号向导页面图

3. 注册

购买注册码，按向导完成注册。运行 QTP，打开 Help 菜单下的 About QuickTest

Professional，出现图 6-3 所示界面。

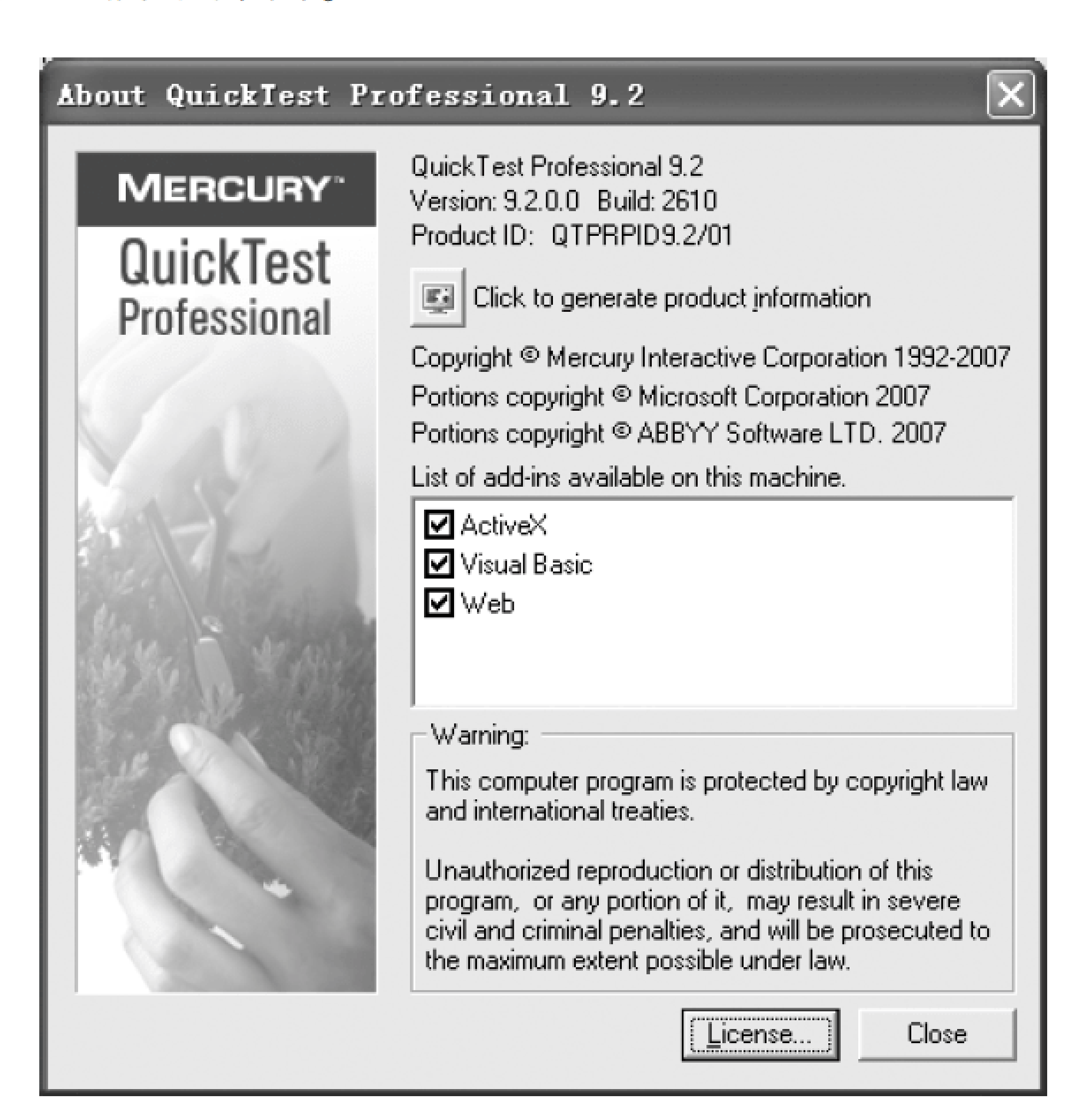


图 6-3 QTP 信息显示图

提示：图中的 ActiveX、VisualBasic 和 Web 是 QTP 自带的内部插件。QTP 在使用其他不同插件时需要选择不同的 License。为提高运行效率，需要有针对性地选择加载所需的插件。

6.3 测试流程

完整的 QTP 测试流程由七个阶段组成[23,25-26]。

1. 准备录制

按测试目标及要求设置应用程序或网站和 QTP。确保应用程序或网站显示要录制的元素，例如，工具栏或指定窗口；确保 QTP 中合理完成录制和运行设置（Automation→Record and Run Settings）和选项设置（Tools→Options），例如，应用程序或 Web 对象模式。

根据 QTP 的功能特点和实现成本，按照测试用例需求或测试文档，决定计划用 QTP 执行的测试和检查的功能点，设计测试数据。

2. 录制应用程序或 Web 上的会话

当使用应用程序或 Web 站点时，QTP 以关键字视图内的一行来显示用户执行的任何引起应用程序或 Web 发生变化的操作步骤，如单击链接或图像，或者在表单中输入数据。

3. 增强测试

通过在测试中插入检查点，可以搜索页面、对象或文本字符串中的特定值，有助于确定应用程序或 Web 站点是否正常运行。

通过参数化，可以扩大测试范围，有助于检查应用程序或 Web 站点如何用数据集来执行同一操作。

通过添加逻辑和条件语句或循环语句，可以向测试中增加复杂的检查。

4. 调试测试

调试测试以确保测试顺利地运行。

5. 运行测试

运行测试，检查应用程序或 Web 站点的行为。在运行时，QTP 将打开应用程序或链接到 Web 站点，并执行测试中的每个步骤。

6. 分析测试结果

检查测试结果以发现应用程序或 Web 站点的缺陷。

7. 报告缺陷

通过 Mercury Interactive 软件测试管理工具 Quality Center，将发现的缺陷报告给数据库。

6.4 Windows 应用程序测试

6.4.1 QTP 主界面

在开始创建测试之前，首先熟悉 QTP 的主窗口，按如下步骤打开 QTP。

(1) 选择 Windows 的"开始"→"程序"→QuickTest Professional→QuickTest Professional 或双击桌面快捷图标 QuickTest Professional，出现 Add-in Manager 对话框，如图 6-4 所示。

提示：如果希望下一次打开 QTP 不出现该对话框，可以清除 Show on startup 复选框，也可以在 QTP 的菜单 Tools→Options 的 General 选项卡内进行设置。

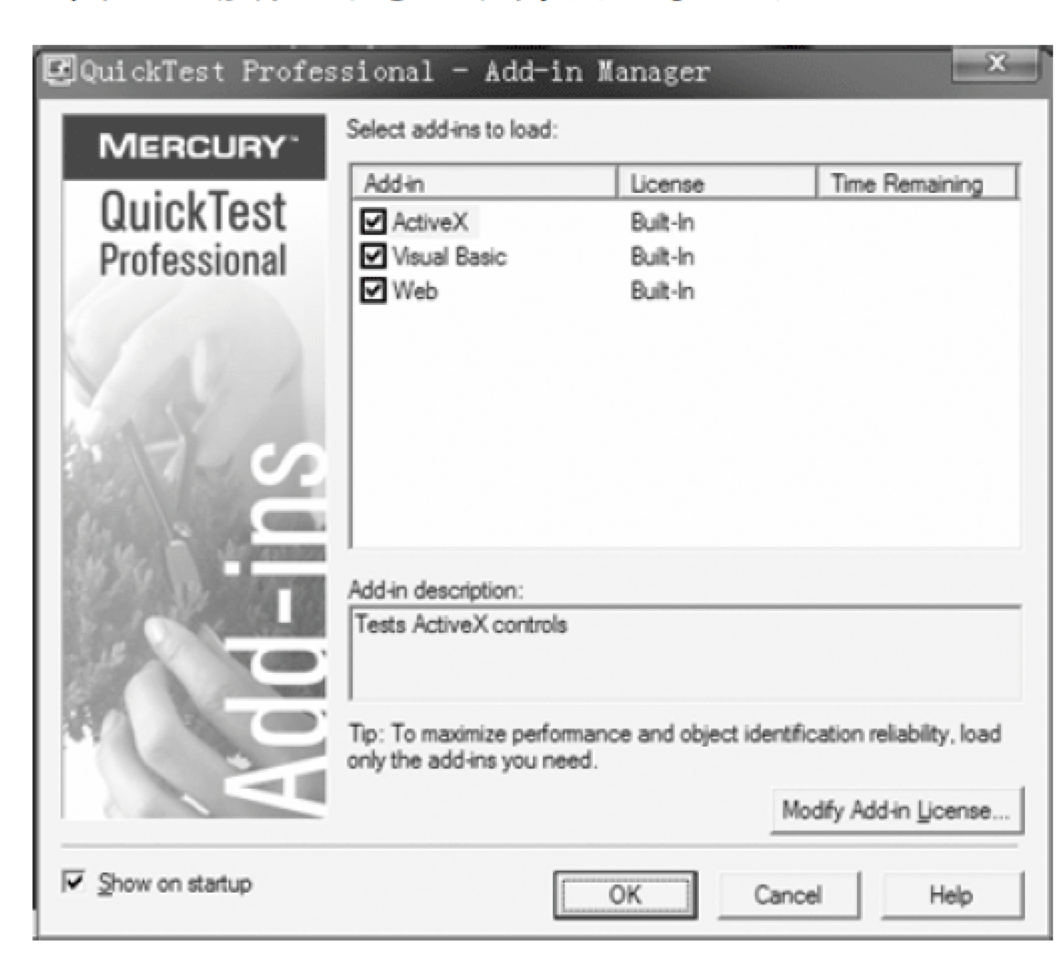


图 6-4 插件管理对话框

(2) 单击 OK 按钮，出现欢迎界面，如图 6-5 所示。在此可以选择教程、开始录制新测试、打开已有测试或新建空测试。若希

望下次打开 QTP 不出现该界面，可以不选择 Show this screen on startup 复选框，或在 QTP 的菜单 Tools→Options 的 General 选项卡内进行设置。

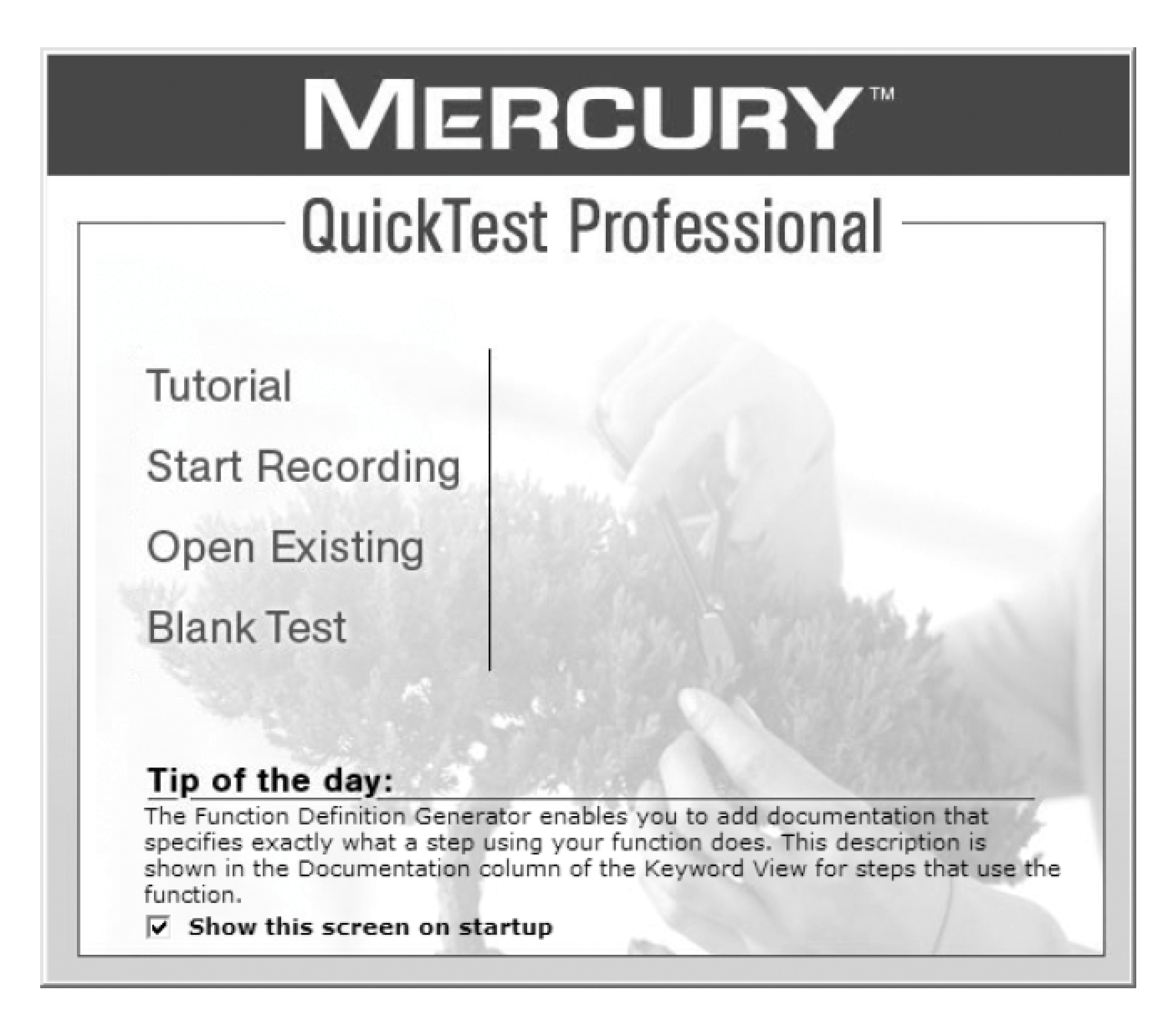


图 6-5 QTP 欢迎界面

当选择创建一个测试(如新建空测试)之后，就显示 QTP 主窗口，如图 6-6 所示。

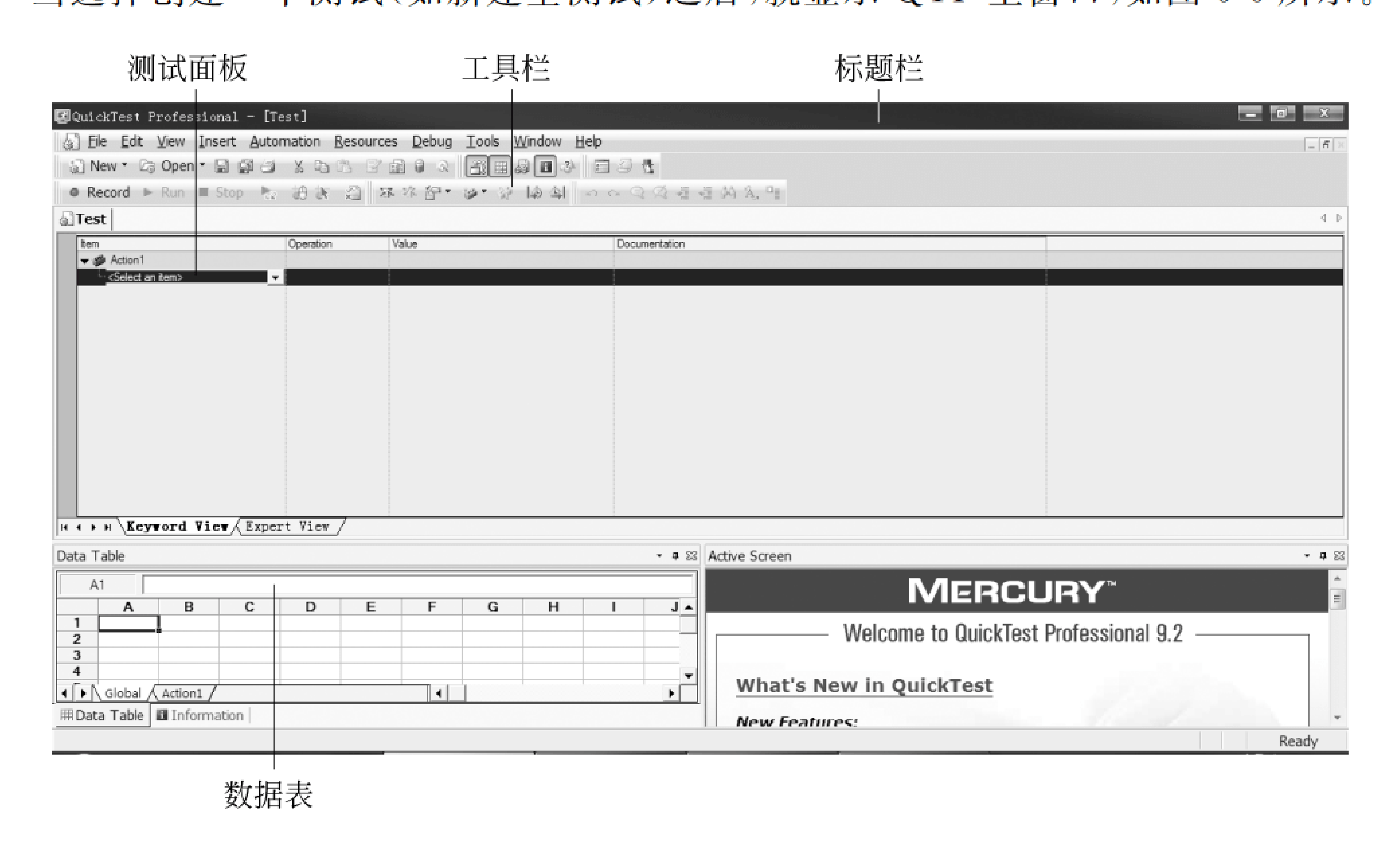


图 6-6 QTP 主窗口

提示：默认打开 QTP 时，主窗口中的有些组件(如调试工具栏、操作工具栏等)不会显示，可以在 View 菜单下选择设置其为显示。其中，操作工具栏只在关键字视图中显示，若在测试中插入可重用操作或外部操作，其将自动显示。

6.4.2 应用程序实例——飞机订票系统 Flight

QTP 软件安装之后，其应用程序中有自带的一个 Windows 版的模拟班机预订与信息服务的应用程序 Flight。其默认安装在绝对路径"C:\Program Files\Mercury Interactive\QuickTest Professional\samples\flight\app"中，其中有三个版本的 Flight，分别为 flight3a、flight4a 和 flight4b，每个版本都有各自的 bug。系统登录密码可通过登录界面的 Help 获取。本章将以 flight4a 版本为例，结合 QTP 完成系统登录及机票预订的测试。

提示：要养成及时查看帮助的良好习惯。

6.4.3 录制测试

当操作执行 Flight 系统时，QTP 可以录制用户的动作步骤。这些动作就是测试的依据。当停止录制时，可以看到关键字视图里新产生的测试步骤。

本节将录制 Flight 系统登录及从 Denver 到 Frankfurt 航班的进程。

(1) 启动 QTP 并新建一个空测试。

(2) 设置录制和运行测试项。选择菜单 Tools→Options 的 General 选项卡，单击 Restore Layout 按钮，以恢复默认主窗口。选择菜单 Automation→Record and Run Settings。确保在 Web 选项卡上选中 Record and run test on any open browser，在 Windows Applications 选项卡上选中 Record and run only on，并选择 Applications opened by QuickTest 和 Applications specified below 复选框。然后，单击绿色"+"号，在出现的 Applications Details(见图 6-7)中添加应用程序可执行文件 flight4a.exe 及其所在路径：C:\Program Files\Mercury Interactive\QuickTest Professional\samples\flight\app。

单击 OK 按钮，完成设置，如图 6-8 所示。

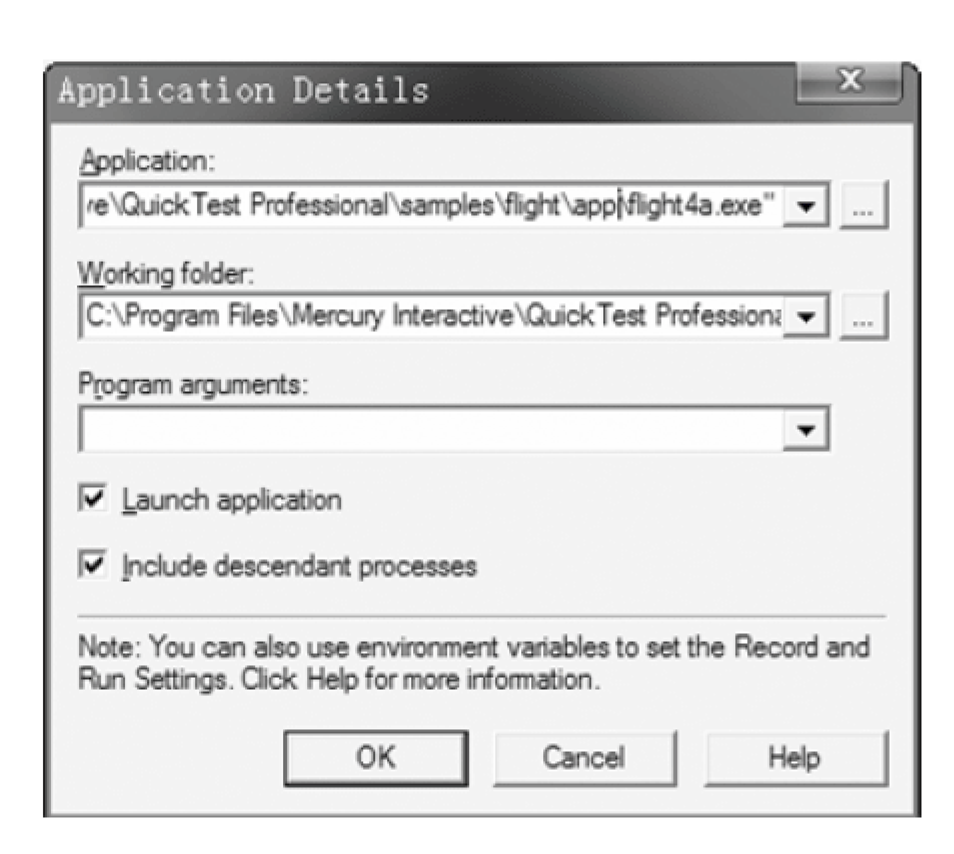


图 6-7 Applications Details 界面



图 6-8 Record and Run Settings 界面

提示：为了后续参数化运行的方便，建议此处选择 Record and run test on any open Windows-based application，并在下一步录制开始后，手工执行 Flight 程序，打开登录界面。

(3) 开始在 Flight 上录制测试。选择 Automation→Record，或单击测试工具栏 Record 按钮，或按 F3 键，QTP 开始录制，且打开 Flight 登录界面。

(4) 登录 Flight。在 AgentName 中输入 mercury，在 Password 中输入 mercury，单击 OK 按钮进入航班预订界面。

(5) 航班预订。在 Date of Flight 中输入时间，如 07/10/11；在 Fly From 列表中选择 Denver；在 Fly To 列表中选择 Frankfurt；单击 Flights 按钮，在 Flights Table 界面中选择所需航班(如第一行 13634 次航班)；在 Name 中输入姓名 wang；选择好机票数量和舱位等级，单击 Insert Order 按钮，生成订单号 21，完成航班预订，如图 6-9 所示。



图 6-9　Flight 系统航班预订界面

(6) 停止录制。选择 Automation→Stop，或单击测试工具栏 Stop 按钮，或按 F4 键，停止录制程序。

至此已预定了一张虚拟的从 Denver 飞往 Frankfurt 航班的经济舱机票。QTP 录制了从 Record 直到 Stop 期间的 Flight 程序所有的操作。

(7) 保存测试。选择菜单 File→Save，取文件名为 FlightTest。确保 Save Active Screen files 复选框被选定。

提示：可通过 View→Expanded All 和 Collapse All 菜单来扩展和收起所有的测试树。

6.4.4　运行测试

选择 Automation→Run，或单击测试工具栏 Run 按钮，或按 F5 键，打开运行对话框，选择默认设置，单击 OK 按钮运行所录制的测试 FlightTest。

当运行测试时,QTP 关键字视图左边距中的箭头指示了正在运行的步骤。正在运行测试的 QTP 界面如图 6-10 所示。

提示:运行测试时,经常会遇到 Object not visible 错误提示,一般都是因为运行时 QTP 界面将应用程序的某一部分界面遮挡住造成的,只需要将应用程序界面移到与 QTP 界面不重复的区域即可。

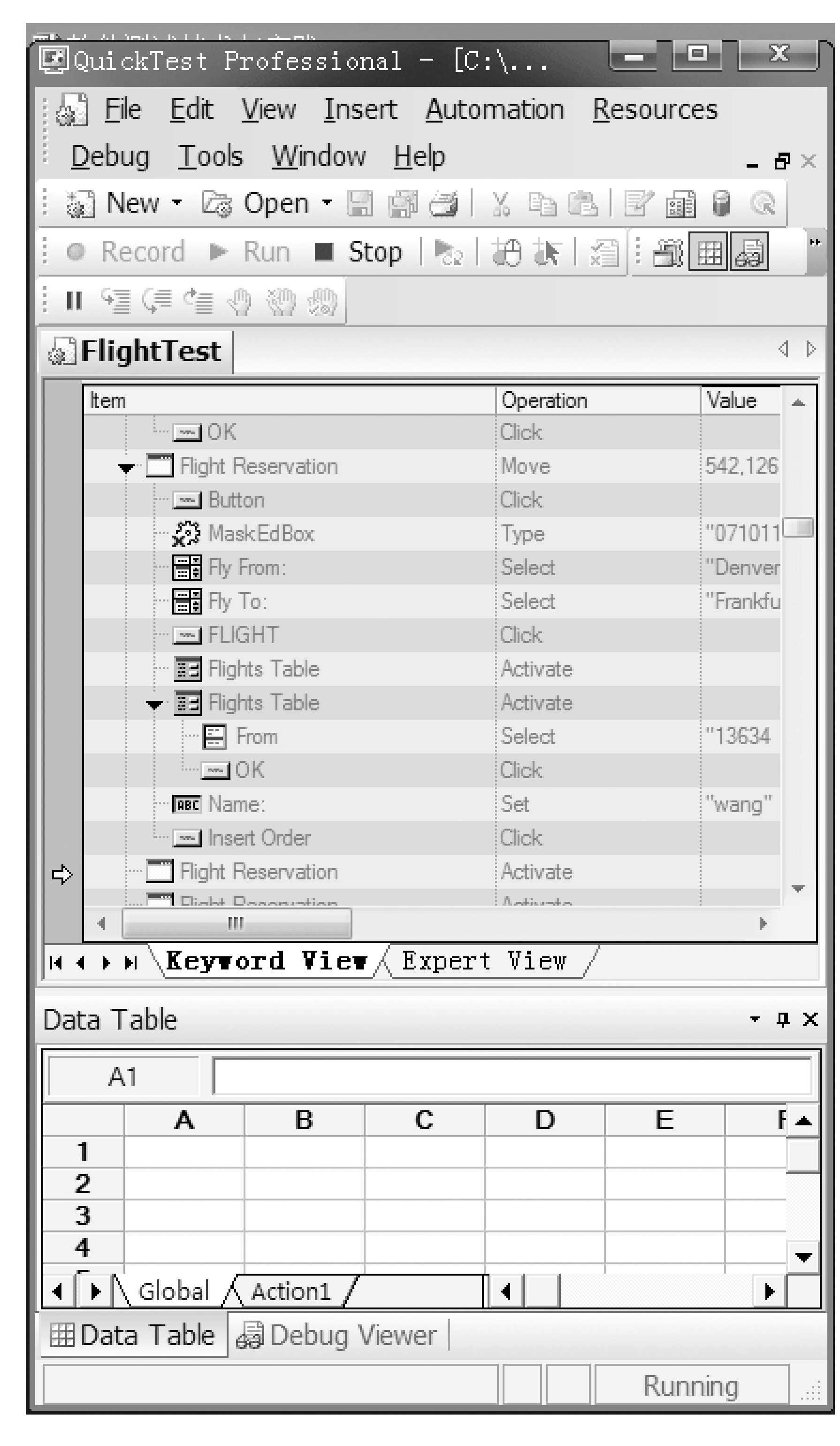


图 6-10 正在运行测试的 QTP 界面

6.4.5 分析测试结果

测试运行结束后,QTP 会自动打开测试结果界面,如图 6-11 所示。

通过测试结果树可以检查运行测试时 QTP 执行的步骤。当单击测试结果树上的某个步骤时,该步骤测试结果的详细信息就会显示在右边面板内,且相应对象被突出显示。

至此,已成功创建并运行了在 Flight 系统上预订从 Denver 飞往 Frankfurt 航班的测试。

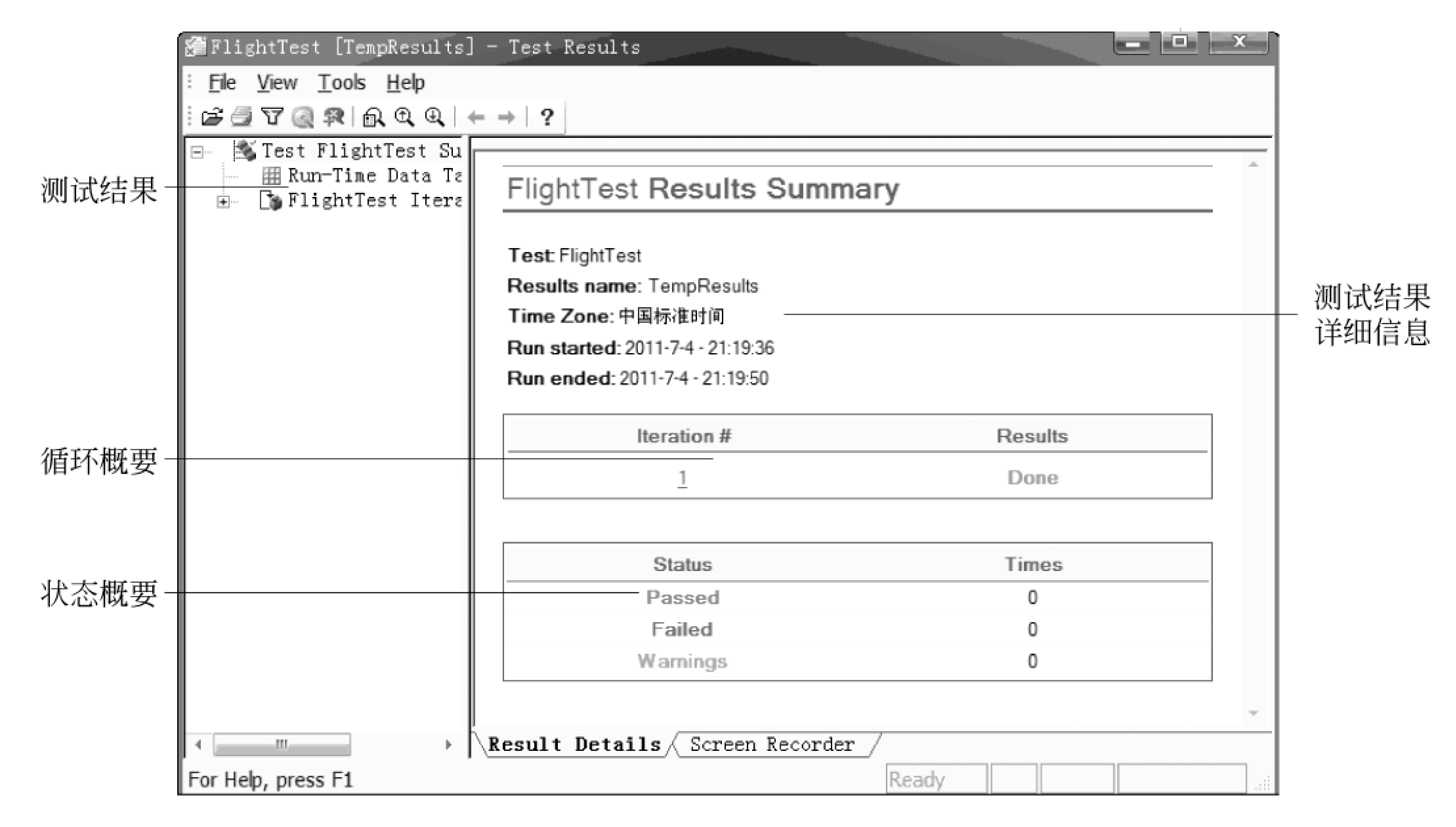


图 6-11 测试结果界面

6.4.6 产生检查点

QTP 提供了多种类型的检查点[23],如表 6-1 所示。

表 6-1 QTP 检查点类型及用法表

检查点类型	描 述	用 法 示 例
标准检查点	检查对象的属性值	检查是否选中某单选按钮
图像检查点	检查图像的属性值	检查图像源文件是否正确
表检查点	检查表中的信息	检查表单元格中的值是否正确
页面检查点	检查 Web 网页的特性	检查 Web 网页载入的时间或者链接是否有效
文本/文本区域检查点	检查文本字符串是否显示在网页或应用程序窗口中的适当位置	检查预期的文本字符串是否显示在网页或对话框上的预期位置
位图检查点	将网页或应用程序的某个区域捕获为位图后对其进行检查	检查网页或网页的任何部分是否能按预期显示
数据库检查点	检查应用程序或网站所访问数据库内容	检查数据库查询中的值是否正确
可访问性检查点	对网站区域进行识别	检查网页上的图像是否包含 ALT 属性(该属性是 W3C Web 内容可访问性规则所要求的)
XML 检查点	检查 XML 文档的数据内容	检查特定的 XML 文件或网页中的 XML 文档
Output Value	检查对象的属性值	检查图像的长度和宽度

提示：

(1) 当 QTP 创建检查点时，它会基于检查点内的信息(如已检查的值)分配名称。即使随后修改了其所基于的信息，检查点名称也不会改变。但 QTP 可能会截短关键字视图中所显示的名称。

(2) 当录制时或录制后，都可以将检查点加入到测试中。

(3) Output Value 检测点在测试运行过程中会将对象的属性值输出到 Data Table 内，运行结束后 Data Table 内数据将清空。

下面以录制的 FlightTest 测试为例，介绍几种检查点的产生，所有操作将在 Active Screen 中完成。

1. 标准检查点

检查 Login 登录框，添加一个标准检查点到测试中。

(1) 打开录制成功的测试 FlightTest。

(2) 选择测试面板测试树中的 Login。

(3) 右击 Active Screen 中的 Login 登录框图片。

(4) 选择 Insert Standard Checkpoint，在出现的 Checkpoint Properties 界面中设置相应的属性，单击 OK 按钮，插入标准检查点。

(5) 运行测试，查看测试报告中检查点运行结果及信息，如图 6-12 所示。

提示：图像检查点也是通过标准检查点产生，当选择图片添加标准检查点时，QTP 会自动将其识别为图像。

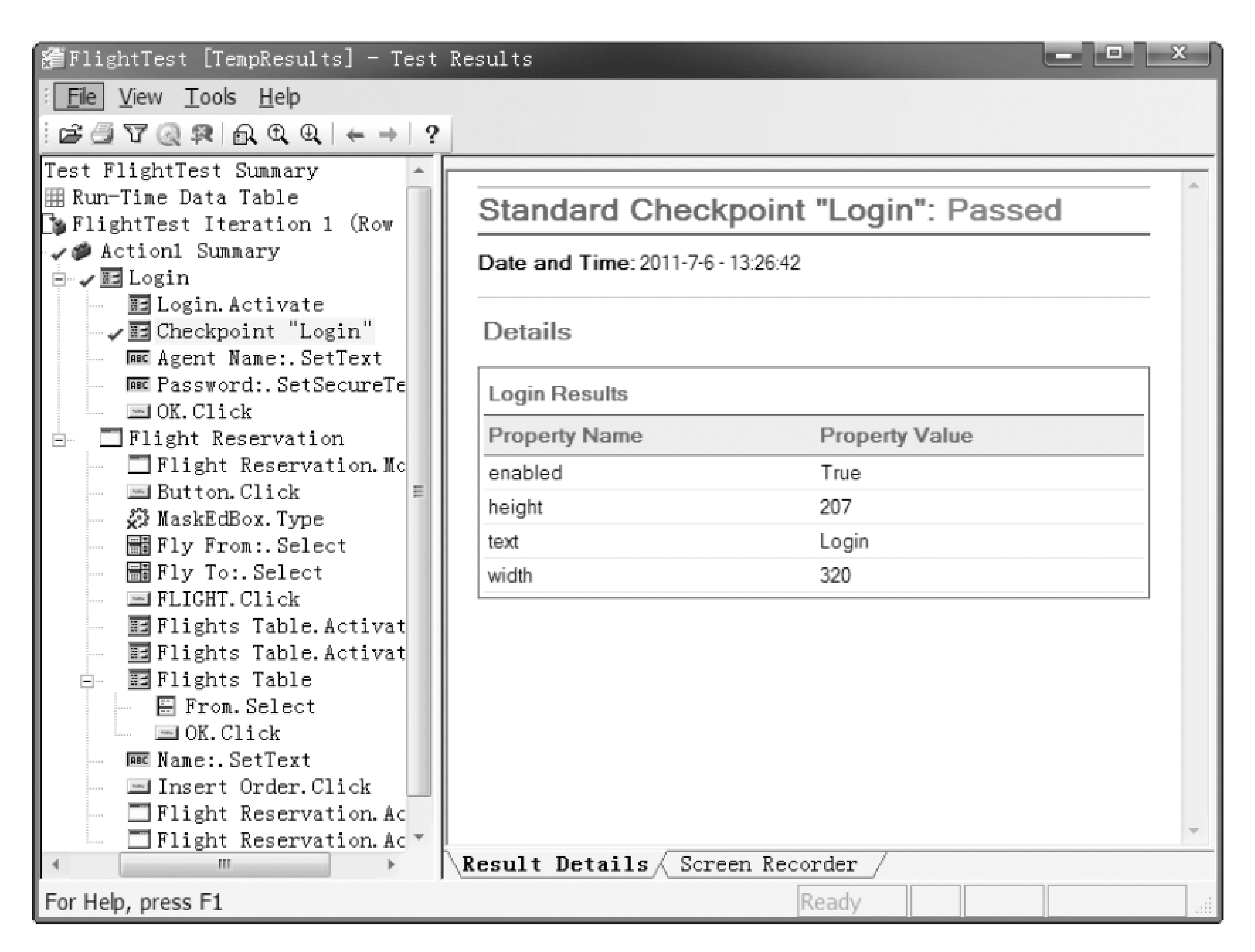


图 6-12　标准检查点测试运行结果及信息

2. 位图检查点

检查 Login 登录界面飞机图片，添加一个位图检查点到测试中。

（1）打开录制成功的测试 FlightTest。

（2）选择测试面板测试树中的 Login。

（3）右击 Active Screen 中的 Login 登录框图片。

（4）选择 Insert Bitmap Checkpoint，出现 Object Selection-Bitmap Checkpoint Properties 界面；单击 OK 按钮，出现 Bitmap Checkpoint Properties 界面；单击 Select Area 按钮，选择飞机位图；单击 OK 按钮，插入位图检查点。

（5）运行测试，查看测试报告中检查点运行结果及信息。

3. 文本检查点

检查 Login 登录界面 Agent Name，添加一个文本检查点到测试中。

（1）打开录制成功的测试 FlightTest。

（2）选择测试面板测试树中的 Login。

（3）右击 Active Screen 中的 Login 登录框的 Agent Name。

（4）选择 Insert Text Checkpoint，出现 Select anObject 界面；单击 OK 按钮，出现 Text Checkpoint Properties 界面；设置好匹配方式后，单击 OK 按钮，插入文本检查点。

（5）运行测试，查看测试报告中检查点运行结果及信息。

4. Output Value

输出 Login 登录界面飞机图片的长度和宽度。

(1) 打开录制成功的测试 FlightTest。

(2) 选择测试面板测试树中的 Login。

(3) 右击 Active Screen 中的 Login 登录框的飞机图片。

(4) 选择 Insert Output Value，出现 Object Selection-Output Value Properties 界面；单击 OK 按钮，出现 Output Value Properties 界面，选择高度和宽度属性；单击 OK 按钮，插入 Output Value。

(5) 运行测试，查看测试报告中检查点运行结果及信息，如图 6-13 所示。

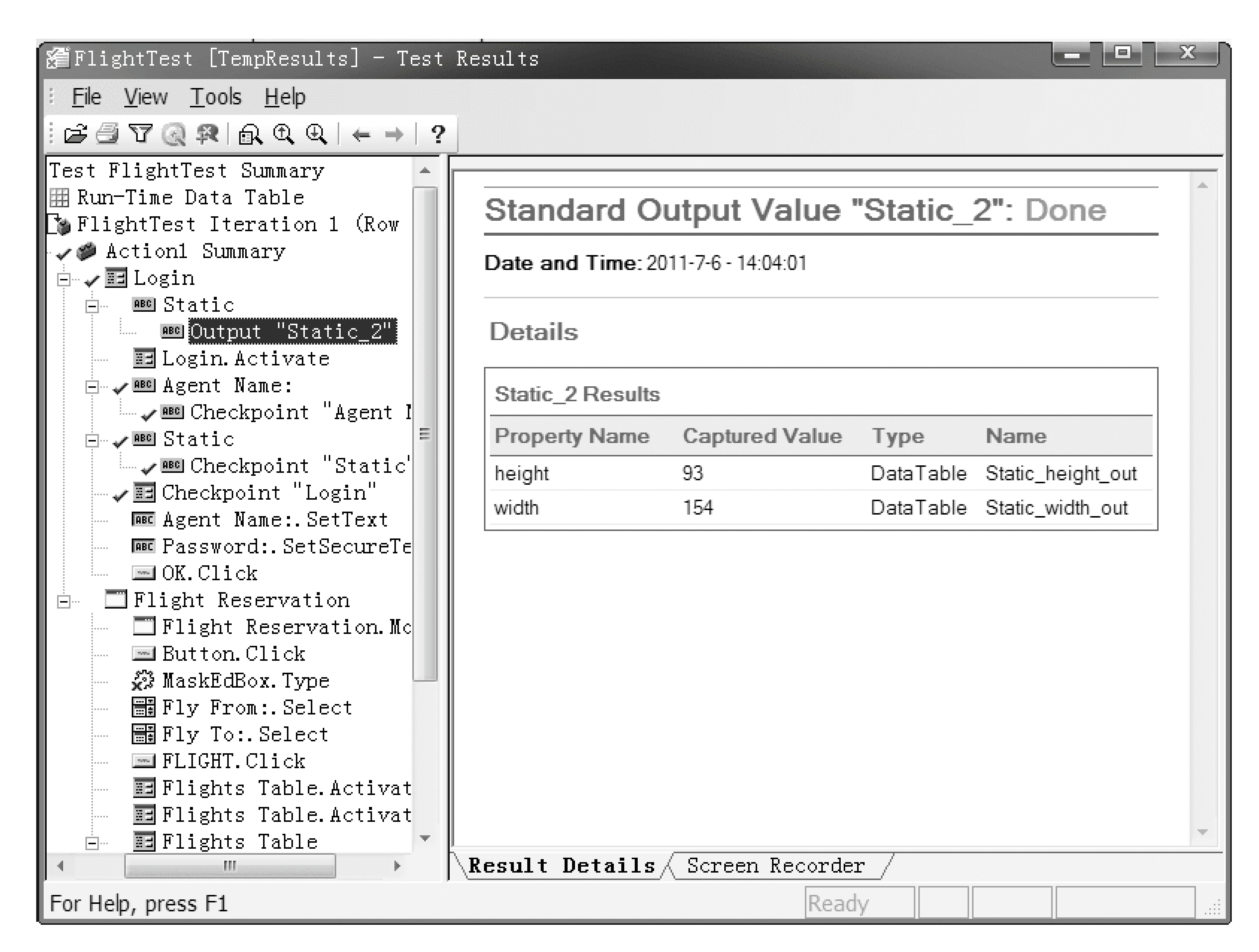


图 6-13 Output Value 输出的长度和宽度属性值

提示：Output Value 检测点在测试运行过程中会将对象的属性值输出到 Data Table 内，运行结束后 Data Table 内数据将清空。

6.4.7 参数化测试

FlightTest 测试中，预定了从 Denver 飞往 Frankfurt 的航班。这里的出发城市和到达城市都是常量，即每次运行测试出发城市都是 Denver，到达城市都是 Frankfurt。本节将使用 DataTable 对预定机票中的 Fly From 和 Fly To 数据进行参数化，以便可以为每个测试运行时使用不同的出发城市和到达城市。

具体步骤如下：

(1) 启动 QTP 及测试。打开录制成功的 FlightTest 测试，如图 6-14 所示。其中，Fly From 和 Fly To 的数据取值都是常量。

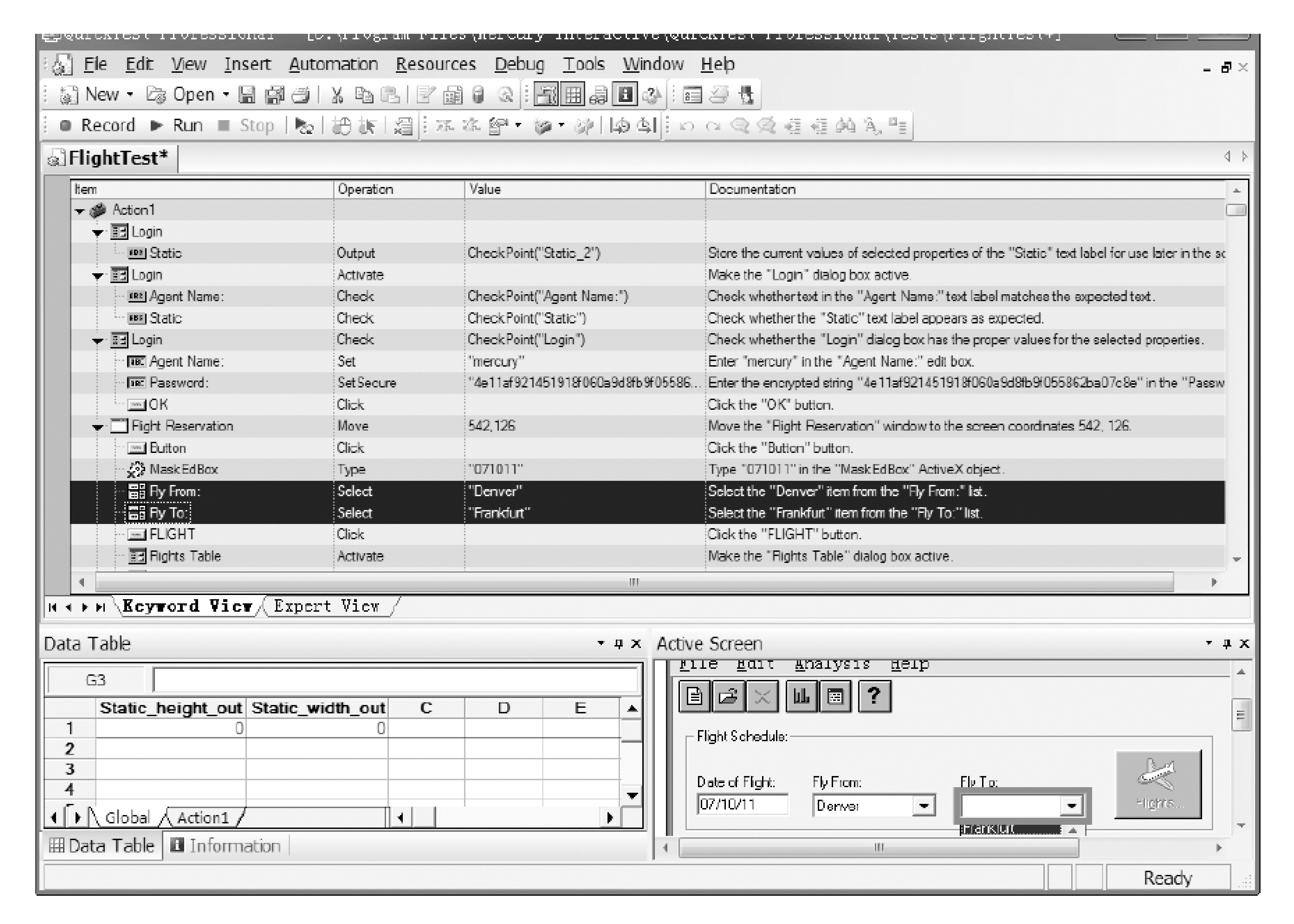


图 6-14 录制成功的 FlightTest 测试

(2) 定义数据表参数并分配参数值。

单击 Fly From 的 Denver 值单元格，单击单元格右上角的 Configure the Value 项。在 Value Configuration Options 界面中选择 Parameter 和 DataTable，在 Name 中定义参数名 fly_from。单击 OK 按钮，在 DataTable 中添加更多的参数取值。用相同的方法参数化 Fly_to。参数定义并赋值的结果如图 6-15 所示。

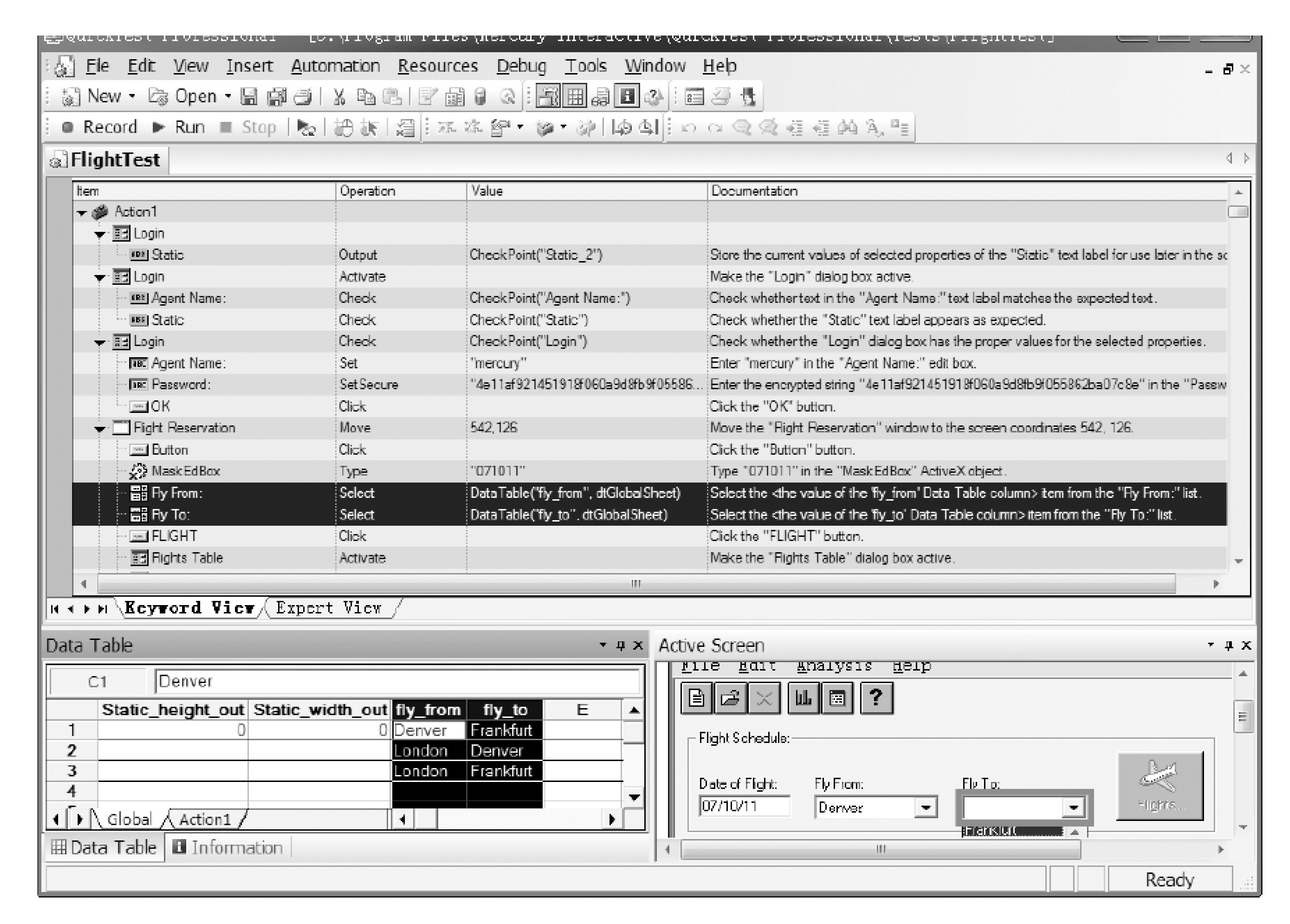


图 6-15 参数定义并赋值的结果

（3）修改参数化测试。

参数化某些测试步骤的值时，其他测试对象可能会受到影响。此时，必须修改这些对象的预期值以匹配从参数化步骤中生成的值。例如，在出发城市和到达城市参数化后，航班班次的选择并没有参数化，导致运行测试运行时航班班次选择出错。因此，必须同时修改航班班次。可以使用 Random Number 技术完成航班班次的选择。

单击 Flights Table 下 From 的值单元格，单击单元格右上角的 Configure the Value 项。在 Value Configuration Options 界面中选择 Parameter 和 Random Number，单击 OK 按钮即可。

（4）运行并分析参数化测试。

运行参数化后的测试，其运行过程如图 6-16 所示。

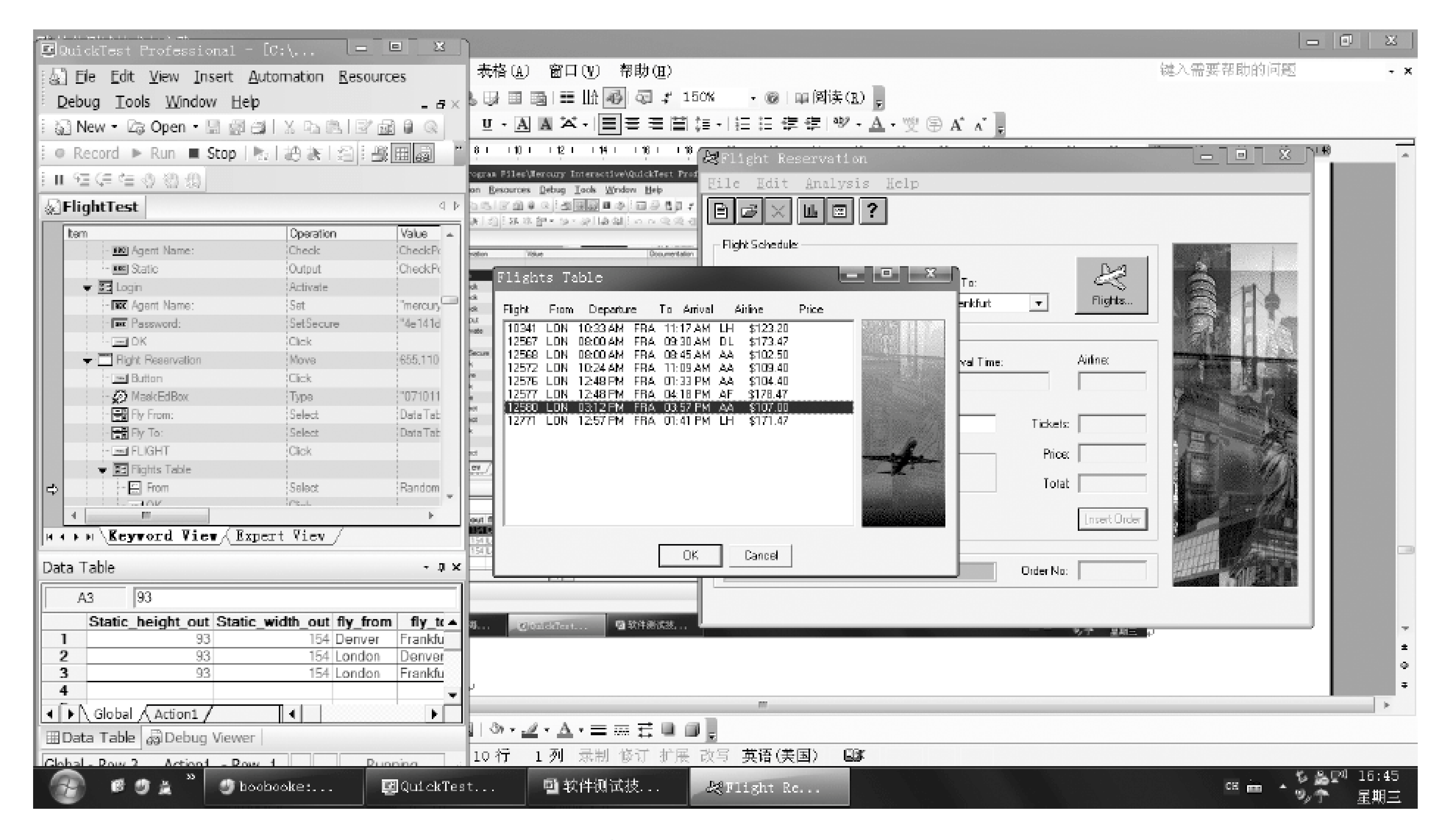


图 6-16　参数化测试的运行过程图

查看参数化测试运行结果，如图 6-17 所示。

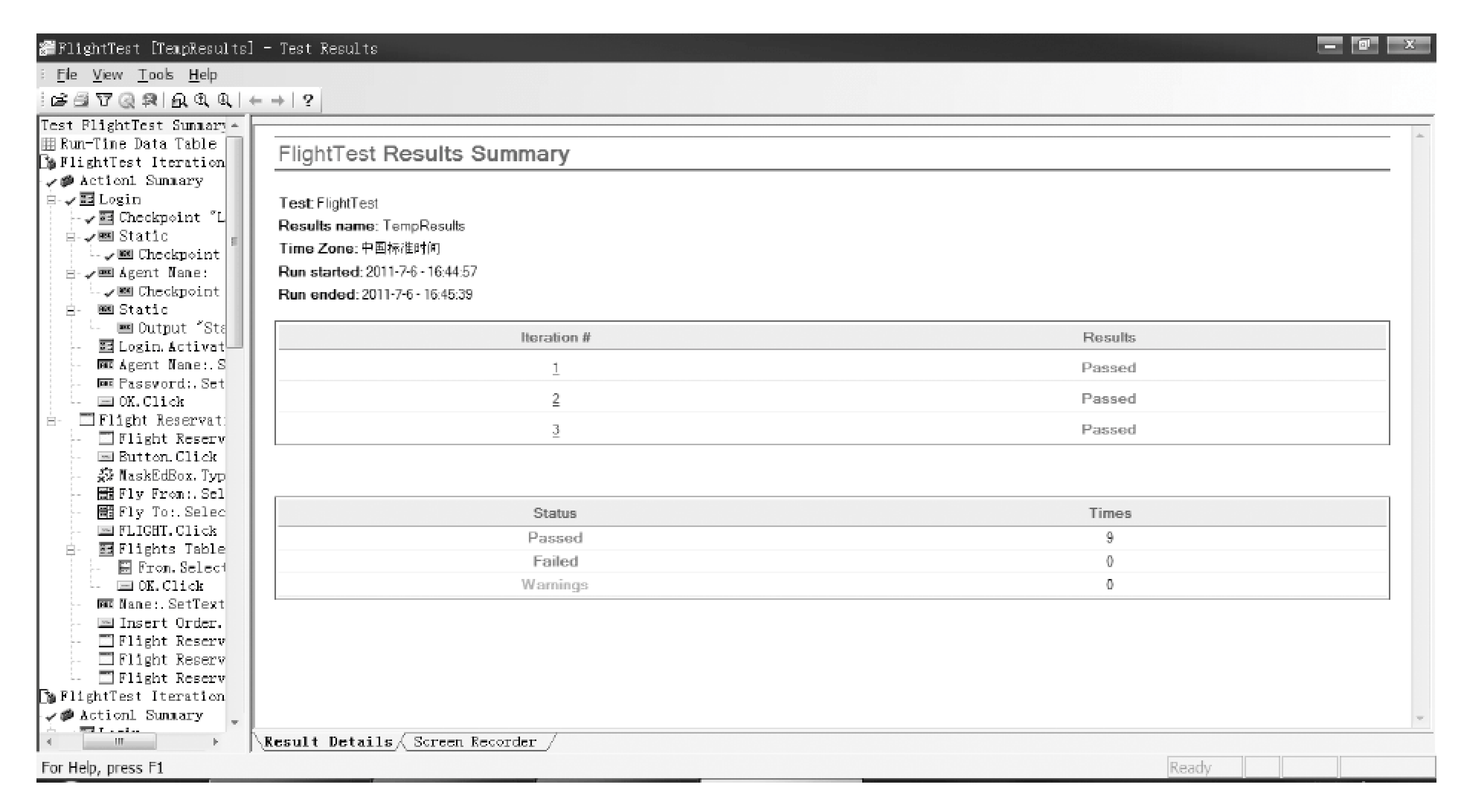


图 6-17　参数化测试运行结果

由图可见，运行共执行了三次，且都成功通过。使用 Run-Time Data Table 显示运行时的数据，如图 6-18 所示。

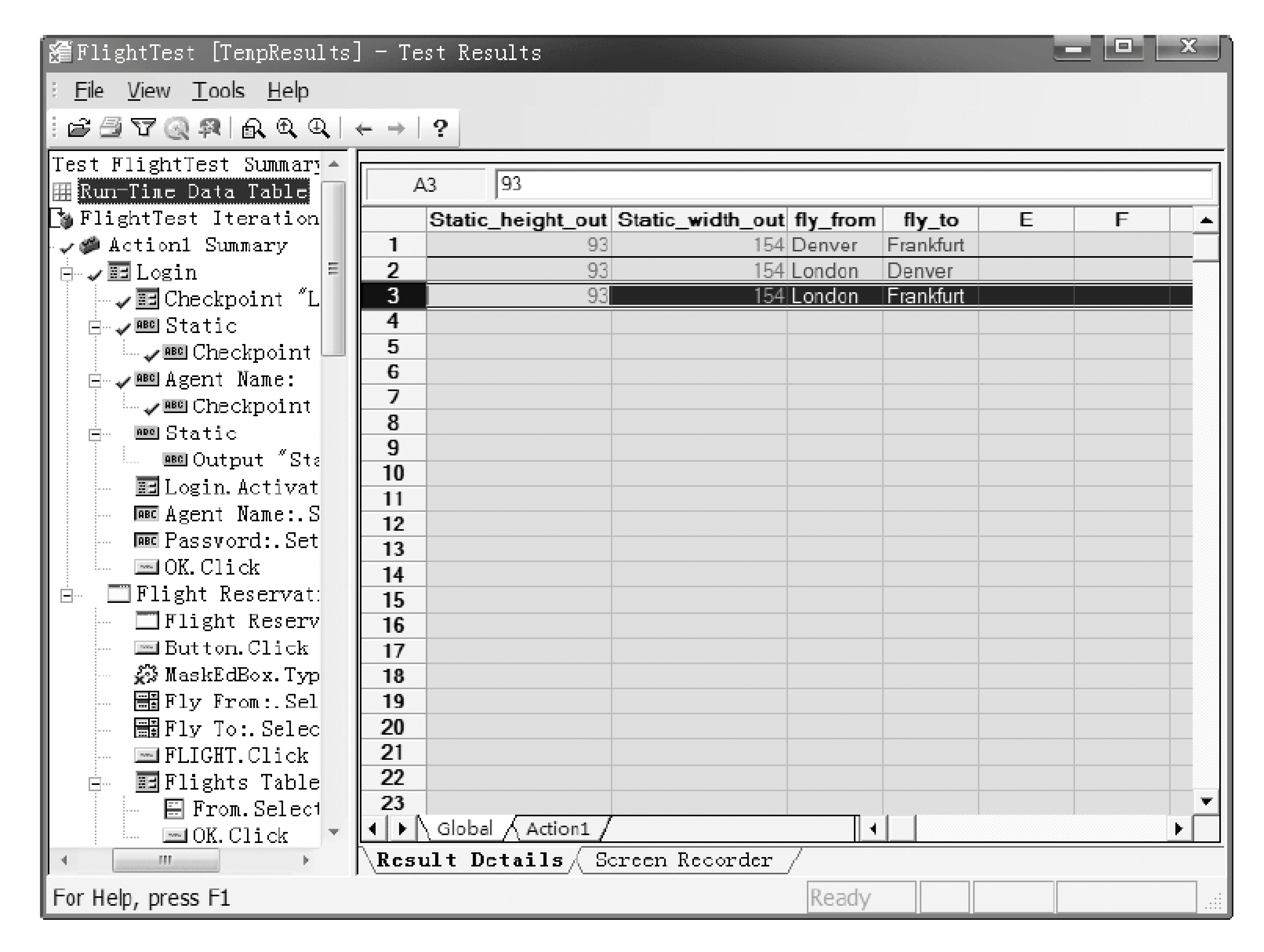


图 6-18　参数化测试运行时的数据图

每次运行的结果及详细信息都可通过测试结果树进行查看分析。

提示：QTP 还可以使用环境变量和 Data Driver 进行参数化，本书不做详细介绍。

6.5　Web 站点测试

本节将以 PHPWind Blog V4.3.10[27] 的登录模块为例介绍使用 QTP 测试 Web 站点。

6.5.1　准备录制

为确保 Web 站点测试的顺利进行，需完成以下设置等准备工作：

(1) 必须确保加载 Web 插件。选择 Help→About QuickTestProfessional 查看。

(2) Record and Run Settings 界面中，选择 Web 选项卡，选择好站点地址和浏览器。同时，在 Windows Applications 选项卡上，选择 Record and run only on，并选择 Applications opened by QuickTest 和 Applications specified below 复选框，确保 Applications Details 空白。该设置可防止录制会话期间无意中录制在各种 Windows 应用程序（如电子邮件等）上执行的操作。

(3) 如果使用 QTP 自带的 mercury tours 网站，需要注意网站的链接地址已经发生了更改。由早期的 http://newtours.mercuryinteractive.com 改为现在的 http://newtours.demoaut.com。

(4) 如果网站上使用了 Java 控件（如 mercury tours 网站的日历选择按钮），需加载 Java 插件（QTP 提供的一种可单独购买的外部插件），否则测试不会录制该操作步骤。

(5) 如果使用 Internet Explorer 作为浏览器，为确保 IE 运行正常，请卸载上网助手

等插件；为保证精确录制所有操作，请在“工具”→“Internet 选项”→“内容”内清除用户名和密码的“自动完成”选项。

(6) 在首次使用 QTP 录制 Web 应用程序测试时，检查浏览器版本是否为 IE6 或 IE7。如果使用的是 IE7.X，需要接受 QTP 对 IE 的设置请求，因为 QTP 不支持多 Tab 浏览器。如果是 IE8，因为 QTP 与 ActiveX 有所冲突，需将 ActiveX 插件不做选择，但若 Web 程序存在 ActiveX 控件时，将会出现无法采集对象的现象。如果默认浏览器是其他的浏览器(如 Tencent Traveler 等)，须在 Record and Run Settings 中选择相应的启动程序，如果选择 Microsoft Internet Explorer，则必须将 IE 设为默认的浏览器，否则将可能会识别为 Windows 对象，无法正常识别 Web 对象，致使一些控件丢失。

(7) 检查浏览器“工具”→“管理员加载项”中的 BHOManager Class 是否为启用，如果为禁用，更改为启用。

(8) 选择 File→Settings，打开 Run 选项卡，可设置运行测试出错时继续执行下一操作步骤；否则，需要人工进行出错处理操作。

(9) 设计测试用例的测试数据。

以测试 PHPWind Blog 登录为例，根据其需求和设计文档，测试设计及测试数据分别如表 6-2 和 6-3 所示。

表 6-2 PHPWind Blog 登录测试设计表

软件名称：PHPWind Blog 系统		**模块名称**：用户功能(登录模块)	
设计者：		**创建日期**：2011-07-20	
设计状态：	**用例类型**：	手工：	**版本号**：V4.0.1
审阅人：	**审阅日期**：		**权重**：
用例描述：			
测试平台：Microsoft WindowsXP			
目的：			
前提条件：1. 使用 test1 为用户名注册为普通用户，密码为 123456 2. 使用 test2 为用户名注册为普通用户，注册后，修改密码为 1234			
测试步骤及输入：			**预期结果**：
正确登录	1. 在“用户名”输入框中输入“正确的用户名” 2. 在“密码”输入框中输入“正确的密码”		1. 系统显示“提示信息”
错误登录	1. 在“用户名”输入框中输入“正确的用户名” 2. 在“密码”输入框中输入“错误的密码”		1. 系统显示“提示信息”
使用未注册用户名登录	1. 在“用户名”输入框中输入“没有注册过的用户名” 2. 在“密码”输入框中输入“错误的密码”		1. 系统显示“没有注册过的用户名”不存在
覆盖需求：			
执行状态： Pass/Fail	**关联缺陷**：		
变更记录：			
变更字段：	**新的值**：	**变更人**：	**变更日期**：

表 6-3 PHPWind Blog 登录测试数据表

No.	正确的用户名	正确的密码	没有注册过的用户名	错误的密码	预期提示信息
1	test1	123456			欢迎您回来
2	admin	1234			欢迎您回来(应该有关于"密码不足 6 位,安全性过低"的提示信息)
3			test	abcdef	test 不存在
4				123456	用户名为空
5	test2				密码为空
6	test1			12abcd	密码错误,您还可以尝试 5 次
7	test1			123abc	密码错误,您还可以尝试 4 次
8	test1			1234aa	密码错误,您还可以尝试 3 次
9	test1			111111	密码错误,您还可以尝试 2 次
10	test1			222222	密码错误,您还可以尝试 1 次
11	test1			333333	已经连续 6 次密码输入错误,您将在 10 分钟内无法正常登录
12	test1	123456			已经连续 6 次密码输入错误,您将在 10 分钟内无法正常登录

6.5.2 录制 Web 上的会话

本节将录制 PHPWind Blog 系统登录。

(1) 启动 QTP 并新建一个空测试。

(2) 设置录制和运行测试项。选择 Tools→Options,打开 General 选项卡,单击 Restore Layout,以恢复默认主窗口。选择菜单 Automation→Record and Run Settings。确保在 Web 选项卡上选中 Open the following address when a record or run session begins,在地址栏中输入地址 http://chen/blog/index.php,在 Open the following browser when a run session begins 中选择可用的浏览器 Microsoft Internet Explorer,如图 6-19 所示。在 Windows Applications 选项卡上选中 Record and run only on,并选择 Applications opened by QuickTest 和 Applications specified below 复选框,确保 Applications Details 为空。

(3) 开始在 Web 上录制测试。

选择 Automation→Record,或单击测试工具栏 Record 按钮,或按 F3 键,QTP 开始录制,且自动打开 PHPWind Blog 首页,如图 6-20 所示。

(4) 录制一个成功的登录。

在用户名中输入 test1,在密码中输入 123456,单击"提交"按钮完成登录。登录成功后,单击"注销"退出登录。

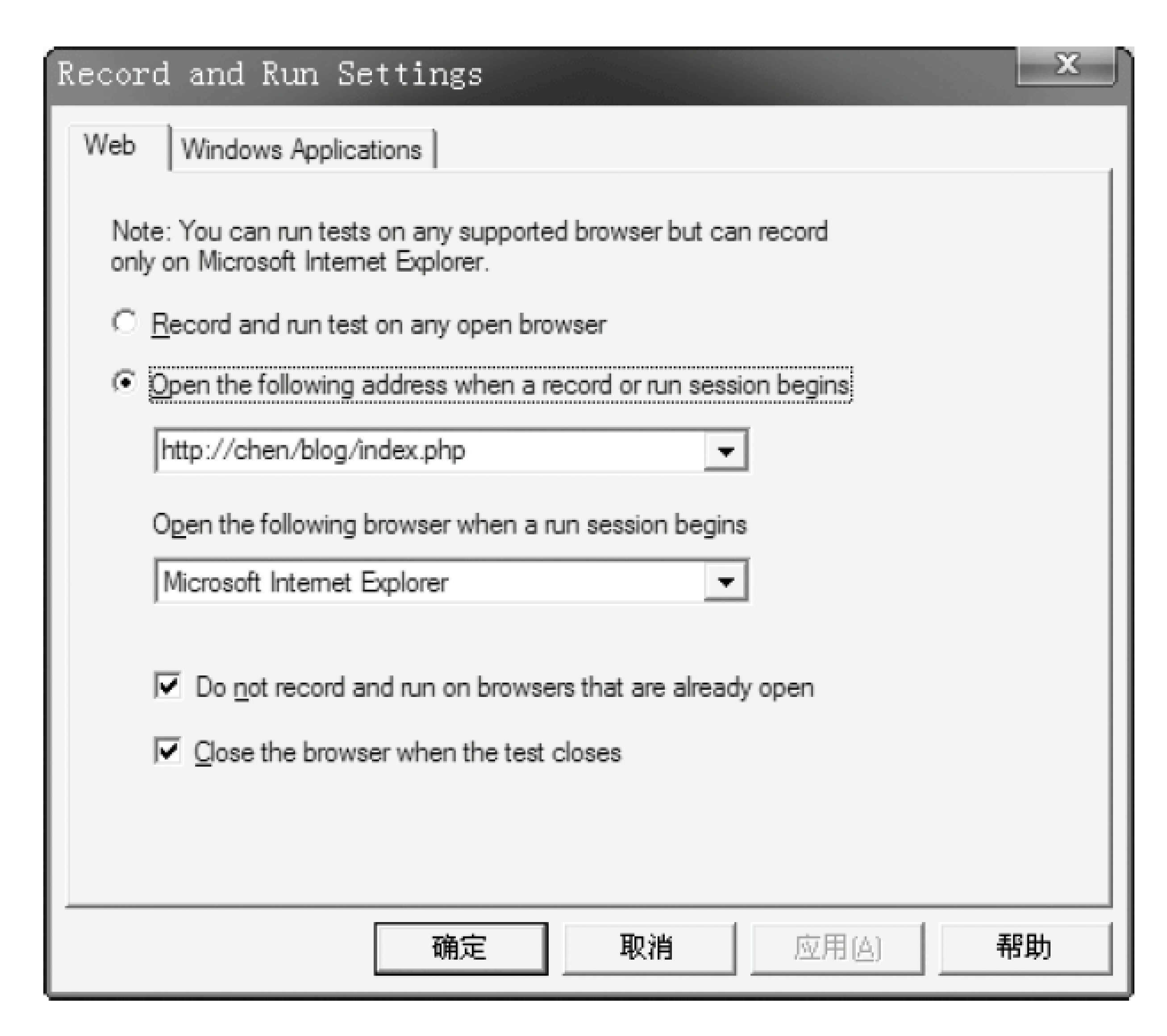


图 6-19 录制 Web 会话设置



图 6-20 PHPWind Blog 首页

（5）录制一个失败的登录。

在用户名中输入 test1，在密码中输入 1234，单击"提交"按钮，出现登录错误提示信息"密码错误，您还可以尝试 5 次"，单击"返回继续操作"按钮返回登录首页。

（6）停止录制。

选择 Automation→Stop，或单击测试工具栏 Stop 按钮，或按 F4 键，停止录制程序。

（7）保存测试。

选择 File→Save，保存文件名为 WebTest。确保 Save Active Screen files 复选框被选定。

6.5.3 增强及调试测试

1. 参数化

根据登录测试数据,需要修改用户名和密码,利用 DataTable 对其参数化。

单击 pwuser 的 test1 值单元格,单击单元格右边的 Configure the Value 项。在 Value Configuration Options 界面中选择 Parameter 和 DataTable,在 Name 中定义参数名 username,单击 OK 按钮。按测试数据,在 DataTable 中添加更多的参数取值。用相同的方法对密码取值参数化,参数名为"userpsw"。

参数化时,可以选择使用 Global Sheet 或 Local Sheet。Global Sheet 参数化的值为全局变量,它控制了整个 Action 的运行次数;Local Sheet 参数化的值为局部变量,只有在本 Action 中可以调用。

2. 增强测试脚本

(1) 在 Expert View 中查看参数化之后的测试脚本如下。

```
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered").WebEdit("pwuser").Set "test1"
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered").WebEdit("pwpwd").SetSecure "4e25807df010f7ea303dfe6d6b818f1777b7"
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered").WebButton("提 交").Click
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_2").Link("注销").Click
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_3").WebEdit("pwuser").Set "test1"
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_3").WebEdit("pwpwd").SetSecure "4e25813bbed01951c2a46c04122d"
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_3").WebButton("提 交").Click
    Browser("PHPWind Blog - powered").Page("PHPWind Blog - powered_4").Sync
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_4").WebButton("返 回 继 续 操 作").Click
```

(2) 在 DataTable 中,双击列名 C,输入列名 status,创建一个预期值列表并按测试数据添加其参数值。

(3) 在登录成功或失败后的信息提示页面的提示信息处,插入一个文本检查点和文本输出点,并将该提示信息存于 DataTable 的 outputtext 列。右击选中提示信息,选择 Insert Text Checkpoint,在 Text Output Value Properties 中,选择 Parameter,单击 Parameter Options,在 Parameter types 下选择 DataTable,在 Name 中输入 outputtext,完成文本检查点添加;右击选中提示信息,选择 Insert Text Output,选择 Output Text,

单击 Modify，在 Parameter types 下选择 DataTable，在 Name 中输入 outputtext，完成文本输出点的添加。

对于登录成功页面的文本检查点和文本输出点，因为提示信息是在登录用户名之后的，而用户名是不断变化的，所以必须将用户名设置为参数形式。在 Text Output Value Properties 中，选择 Text Before，选择 Parameter，单击 Parameter Options，在 Parameter types 下选择 DataTable，在 Name 中输入 username。

(4) 在测试脚本中，定义变量 istatus 并赋值为测试用例设计的预期提示信息。

```
Dim istatus'定义变量 istatus
istatus= DataTable("status", dtGlobalSheet)'读取预期提示信息并赋值给 istatus
```

(5) 增加条件控制。根据测试数据的预期提示信息，使用 QTP 的条件控制语句设计测试脚本，确保测试顺利地多次重复执行。

(6) 调试测试脚本。

经增强调试后的测试脚本如下。

```
    Dim istatus'定义变量 istatus
    Dim Lpwd'定义密码长度变量
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered").WebEdit("pwuser").Set DataTable("username", dtGlobalSheet)
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered").WebEdit("pwpwd").SetSecure DataTable("userpsw", dtGlobalSheet)
    Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered").WebButton("提 交").Click
    istatus= DataTable("status", dtGlobalSheet)'获取预期提示信息
    Lpwd= DataTable("userpsw", dtGlobalSheet)'获取密码
    If istatus= "欢迎您回来" Then  '判断是否登录成功
        Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_6").Output CheckPoint("PHPWind Blog - powered_2")
        Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_6").Check CheckPoint("PHPWind Blog - powered_6")
        Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_2").Link("注销").Click
        If len(Lpwd)< 6 Then   '密码过短
        reporter.ReportEvent micWarning,"登录","登录成功,但密码少于 6位"   '密码
低于 6位顺利登录时,测试报告中给出警告
        end if
    else
        Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_5").Check CheckPoint("PHPWind Blog - powered_4")'添加文本检查点
        Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_5").Output CheckPoint("PHPWind Blog - powered_5")'添加文本输出
        reporter.ReportEvent  micPass,"登录","登录失败,给出提示信息,具体参见文本检测点及
文本输出内容"'未能成功登录时,测试报告中给出提示信息
```

```
        Browser("PHPWind Blog - powered").Page("PHPWind Blog -
powered_4").WebButton("返 回 继 续 操 作").Click
    end if
```

至此，PHPWind Blog 系统的登录测试 WebTest 建立完成，其 Keyword View 和 Expert View 视图分别如图 6-21 和 6-22 所示。

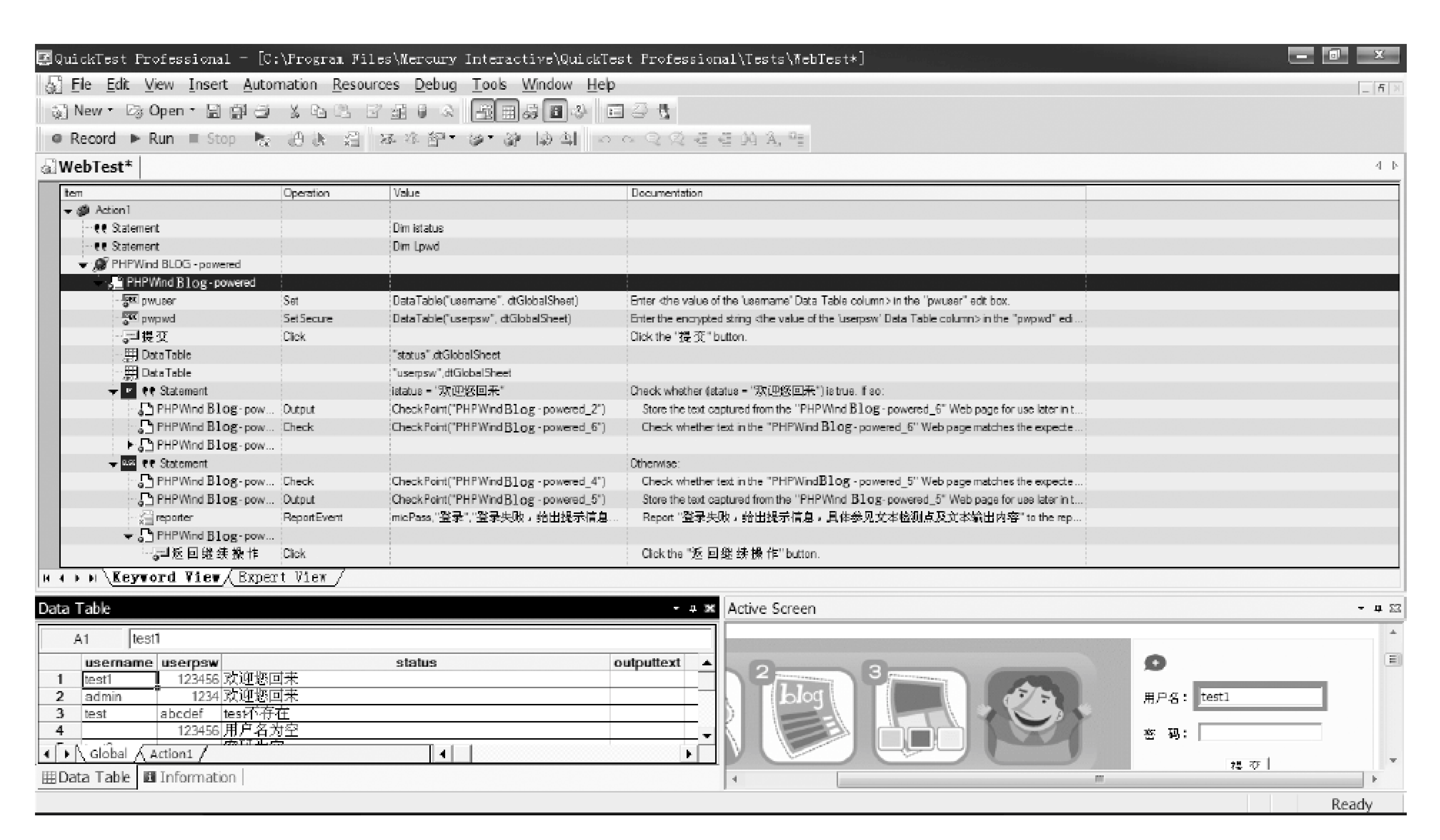


图 6-21　所建立 WebTest 测试的 Keyword View 视图

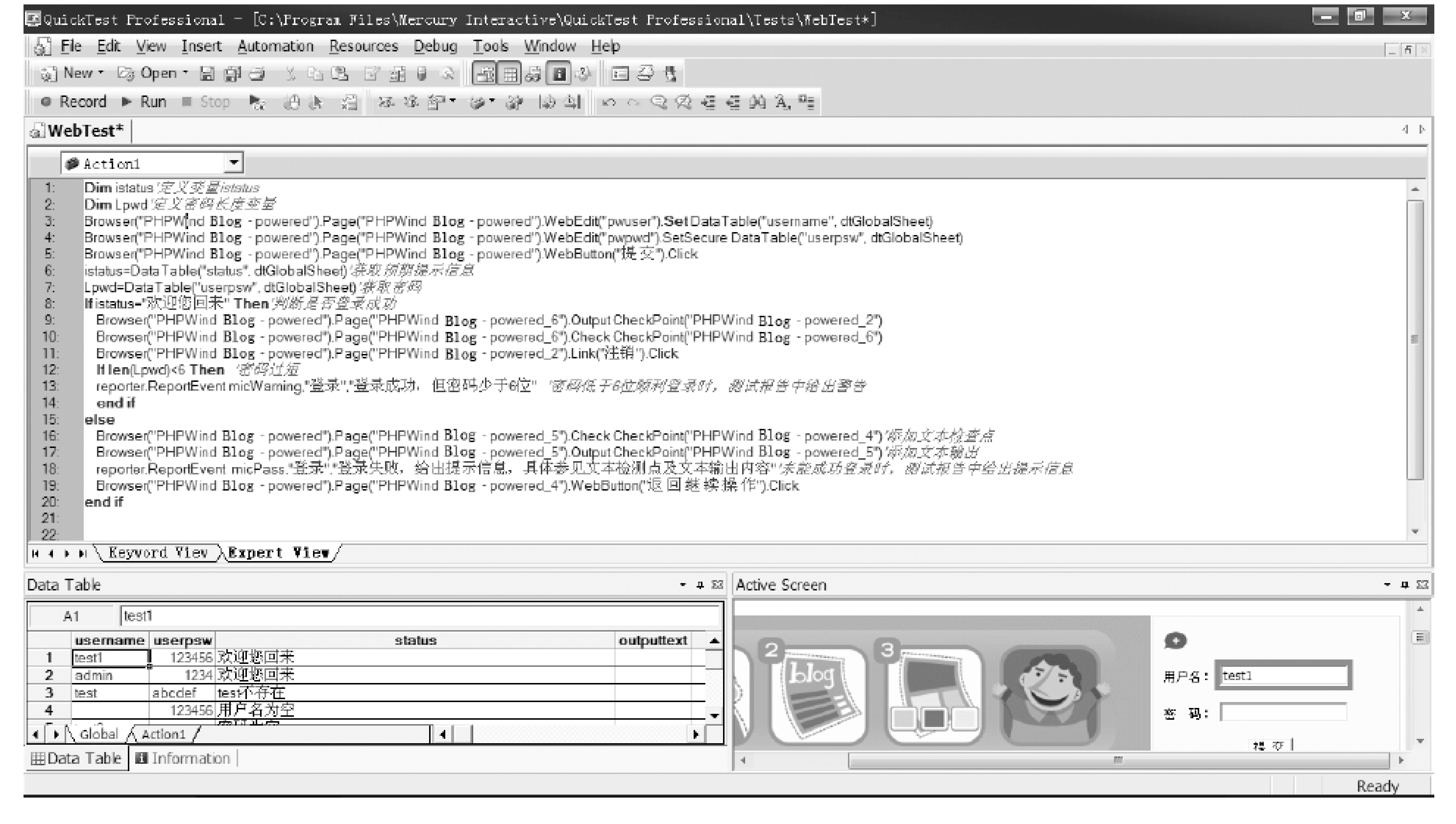


图 6-22　所建立 WebTest 测试的 Expert View 视图

6.5.4　运行测试

选择 Automation→Run，或单击测试工具栏 Run 按钮，或按 F5 键，打开运行对话框，

选择默认设置，单击确定按钮运行所录制的测试 WebTest。

6.5.5 分析测试报告并提交缺陷

测试运行结束后，QTP 会生成测试报告，如图 6-23 所示。测试报告中有“×”或者“!”标记的报告项，可能是执行脚本出错或者检查点校验没有通过，可能是一个缺陷。

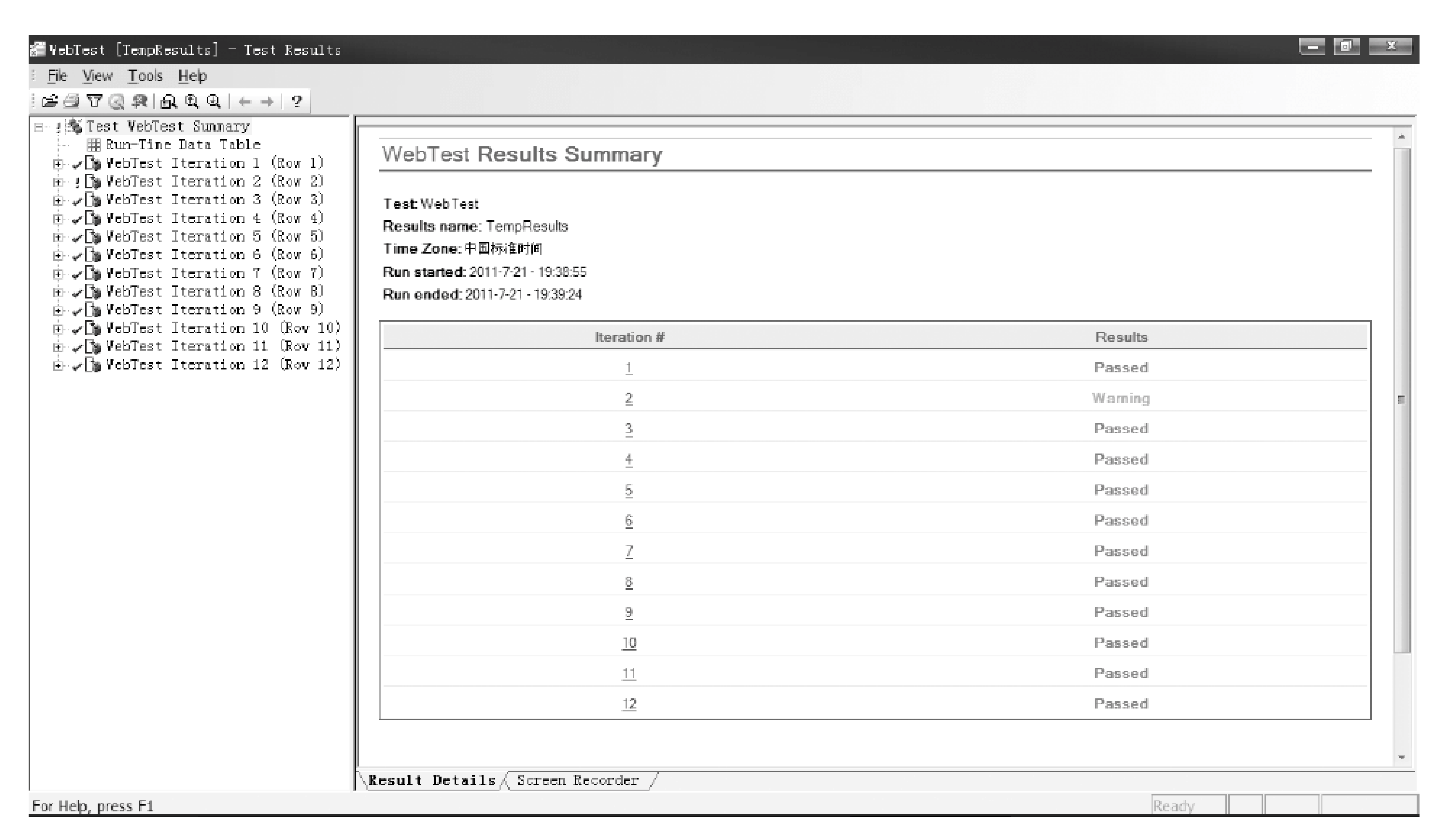


图 6-23 WebTest 测试报告

由报告可见，共循环执行 12 次登录。其中，第 2 次存在警告信息，其余测试都顺利通过。运行测试的实时数据如图 6-24 所示。

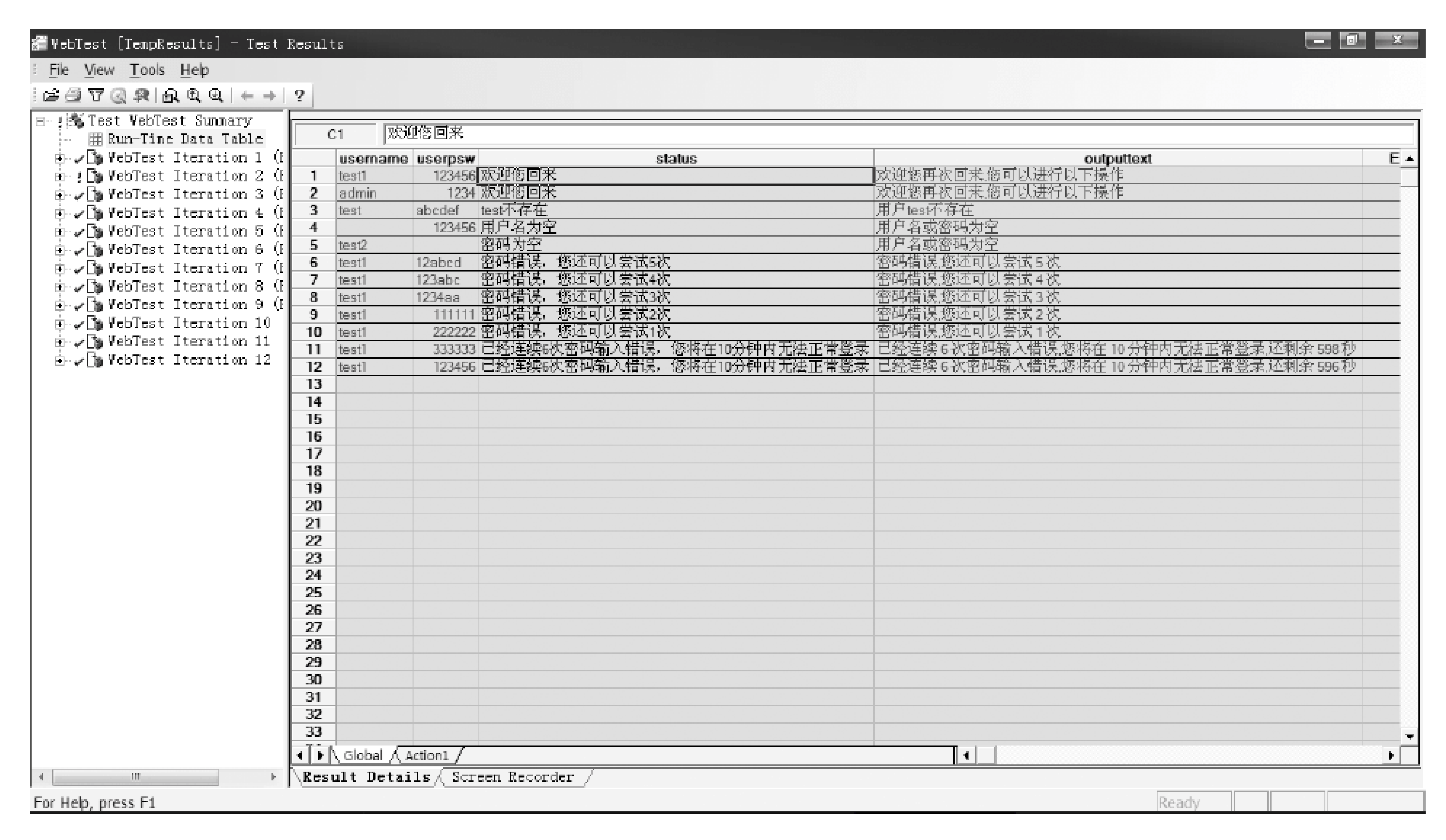


图 6-24 测试实时数据

由图可见，预期提示信息和系统实际提示信息基本相同，登录功能基本实现。未注册(不存在)用户 test 登录时，系统能正常提示“用户 test 不存在”。用户名或密码有一处

为空时，系统能正常提示“用户名或密码为空”。同一用户短时内多次输入密码错误时，系统能正常提示密码错误，且能正常控制的重复次数为 6 次。当用户超过 6 次登录失败时，无论接下来输入错误还是正确的密码，系统都将提示用户已经连续输错密码超过 6 次，请在第 6 次登录失败后的 10 分钟之后再登录，且提示用户还需再等待多长时间。

按照检查点的设置，不同用户登录成功后应检查该用户之后的提示信息。运行测试时，test1 登录成功并注销后，再用 admin 登录，成功登录后应显示 admin 后的提示信息。由图 6-25 可见，文本检查点采集了 admin 和“我的博客 博客”之间的“欢迎您再次回来，您可以进行以下操作”信息，表明检查点检查成功。

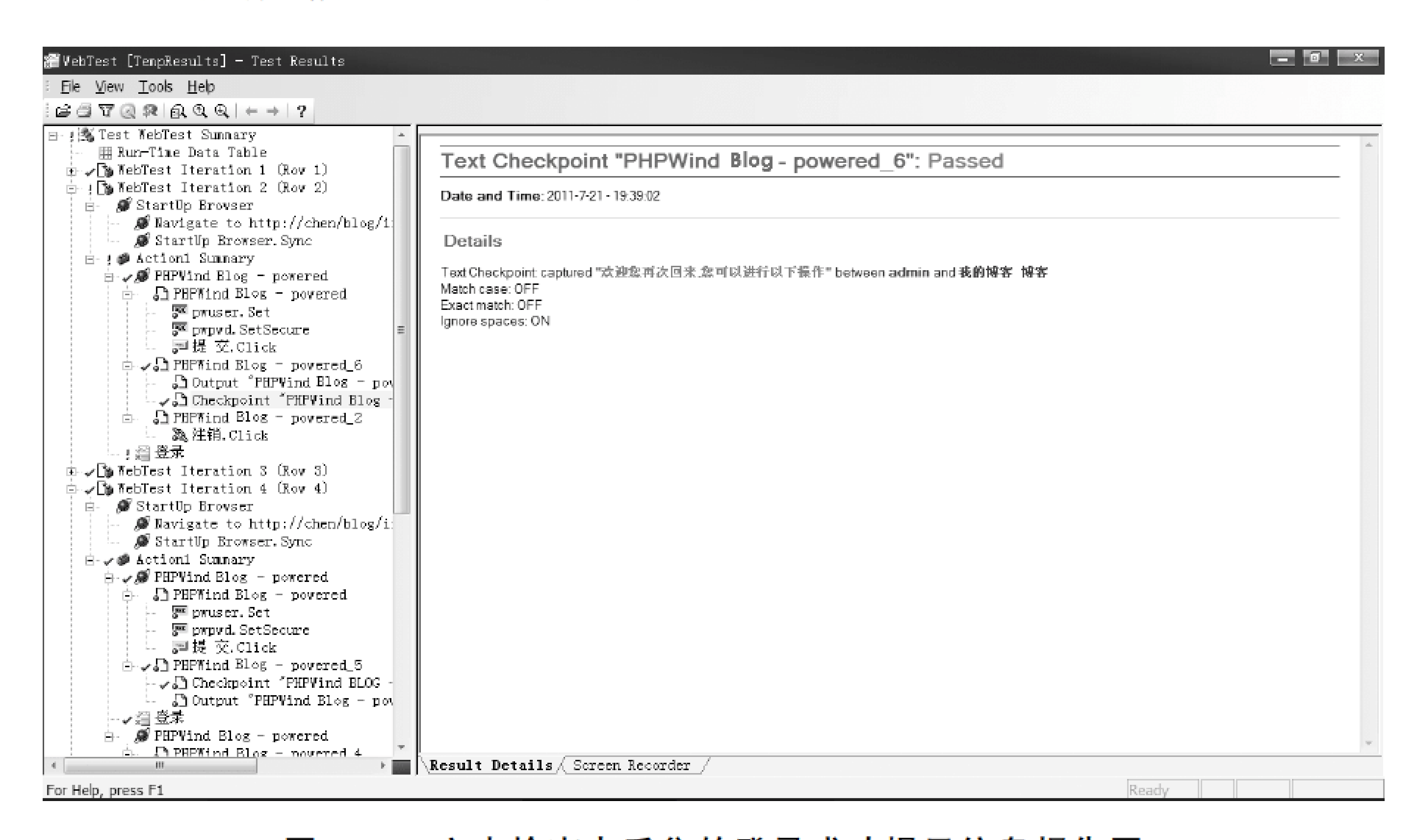


图 6-25　文本检查点采集的登录成功提示信息报告图

同时，按照用户注册时系统要求密码不能低于 6 位。注册登录后，可成功将密码修改为低于 6 位。admin 密码修改为 4 位 1234，经测试，可成功登录。根据测试脚本，密码长度低于 6 位时，报告中添加警告信息“登录成功，但密码少于 6 位”，如图 6-26 所示。另



图 6-26　密码长度低于 6 位时的警告提示信息报告图

外，按照设计用例，此时系统成功登录的提示信息中应该包含“密码低于 6 位，安全性过低”的信息。但由图 6-25 可见系统没有任何关于密码安全性方面的提示信息，因此，此可视为系统登录模块的一个缺陷。

根据系统功能，同一用户登录时，短时内连续 6 次输错密码，将在最后一次登录失败后的 10 分钟内不能正常登录。测试中，test1 连续 6 次密码错误后，再使用正确的密码 123456 进行登录，系统提示“已经连续 6 次密码输入错误，您将在 10 分钟内无法正常登录，还剩余 596 秒”，如图 6-27 所示。

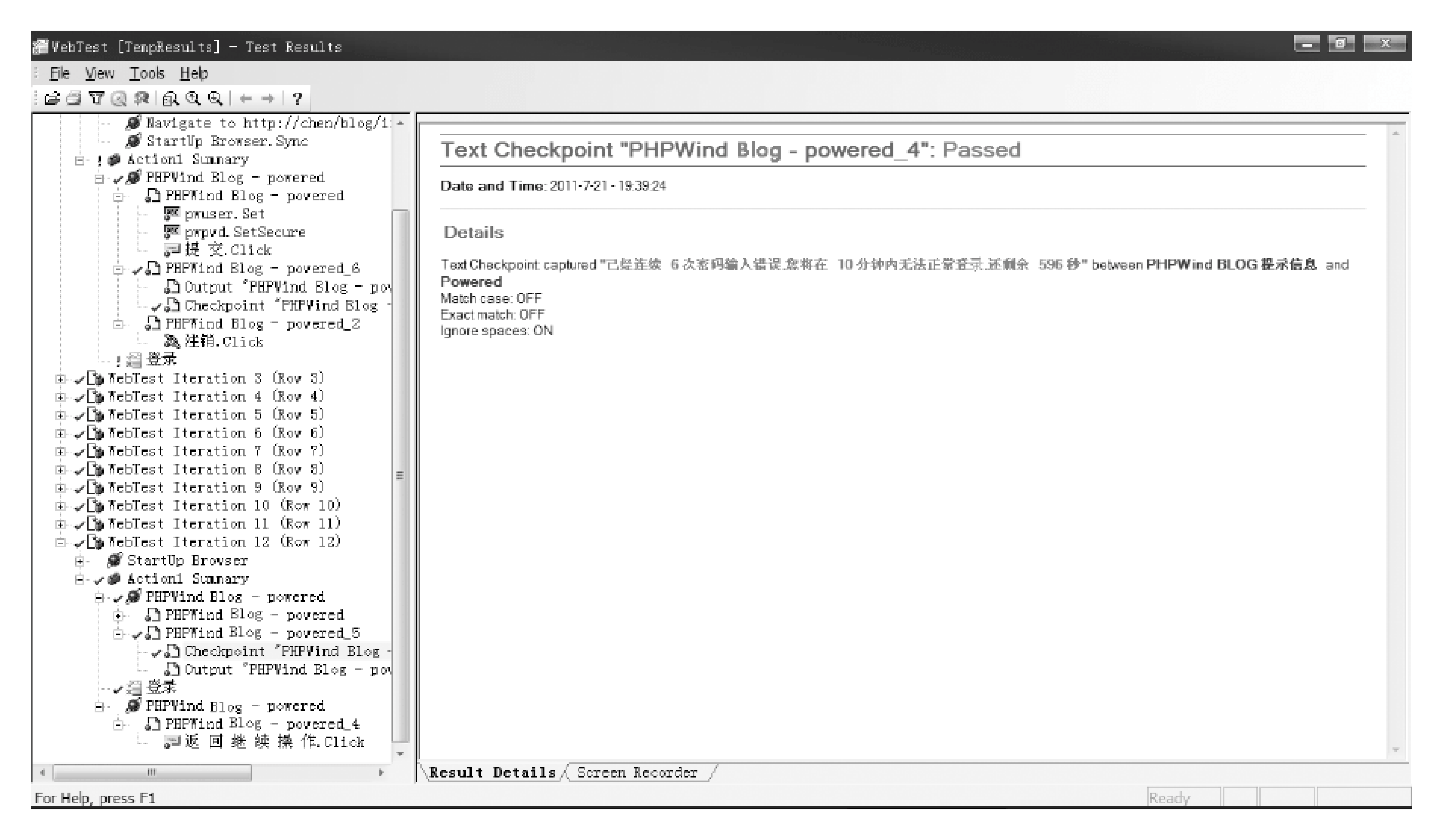


图 6-27 连续 6 次密码输错后再登录时系统提示信息报告图

根据系统功能，登录失败时，系统给出提示信息，并可点击“返回继续操作”按钮返回到登录首页。按测试数据第 4 行，用户名为空，密码为 123456 登录时，系统显示“用户名或密码为空”，如图 6-28 所示。虽然本次测试中，登录失败，但系统的提示信息及功能都与预期的结果完全相同，所以本次测试也是正常通过的。

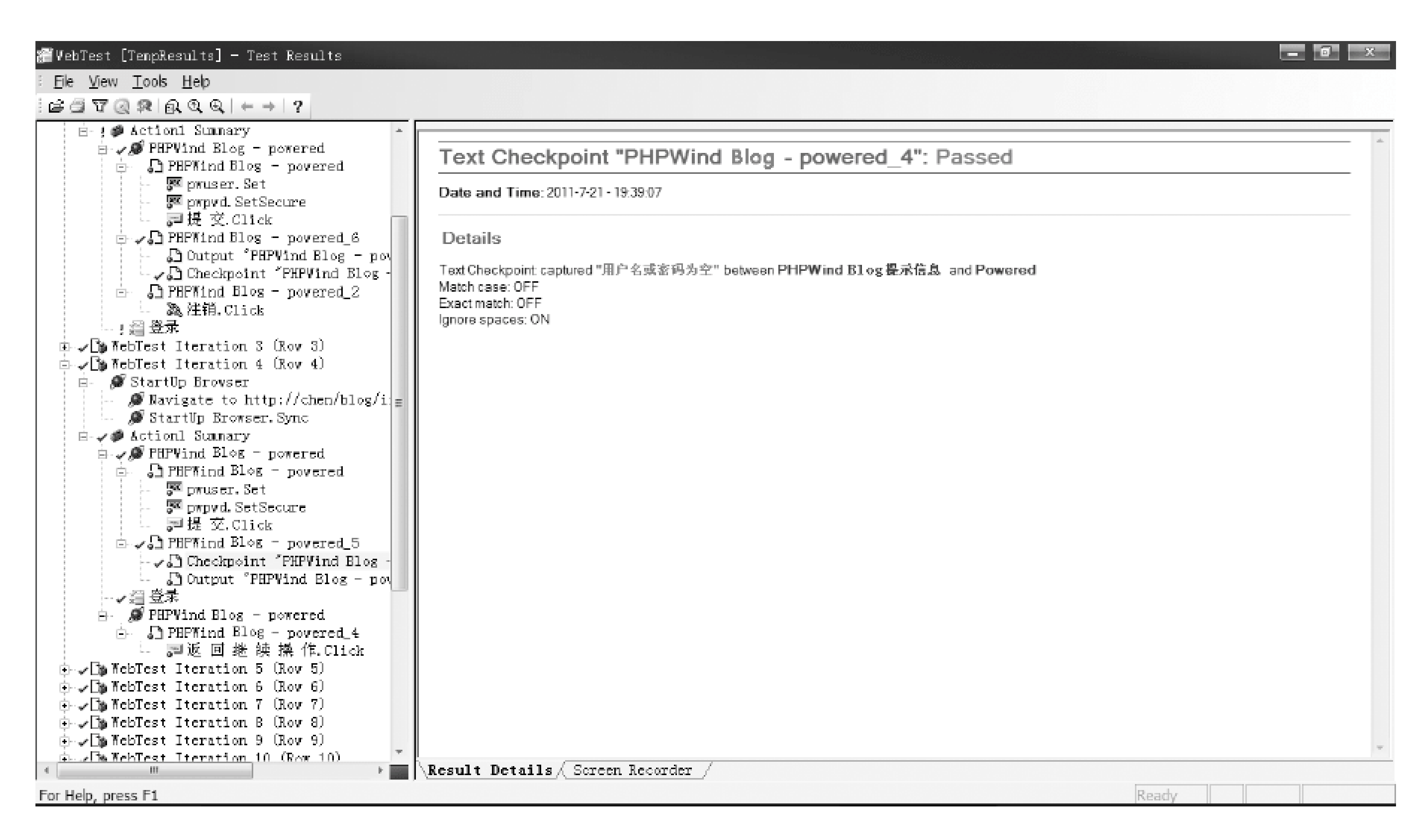


图 6-28 用户名为空登录时的提示信息报告图

通过对测试报告的分析，发现 PHPWind Blog 系统登录模块存在一处缺陷，缺陷记录如表 6-4 所示。

表 6-4　PHPWind Blog 登录模块测试缺陷记录表

用例编号：		**状态：**	New	
产品名称：PHPWind Blog		**产品版本：**	V4.0.1	
模块名称：用户登录模块		**模块版本：**	V4.0.1	
测试者：		**测试日期：**	2011-07-21	
类型： 1. 缺陷 √	2. 安全问题____	3. 功能建议____	4. 界面建议____	
严重级别：1. 紧急____	2. 很高____	3. 高____	4. 中等 √	5. 低____
描述：				
可重现否：Yes				
重现步骤： 1. 按照测试用例数据输入用户名和密码。 2. 登录成功，点击注销；登录失败，点击返回继续操作。				
隔离方法：				
可能存在隐患： 密码修改为低于 6 位时，仍可以成功登录，且没有任何关于安全性过低的提示信息。				
变更记录：				
变更字段：	**新的值：**	**变更人：**	**变更日期：**	

第 7 章



面向对象的单元测试

7.1 面向对象的单元测试

单元测试是测试过程中的最小模块，执行过程中依据程序框架对产品和模块进行测试。在面向对象软件时，封装驱动类和对象的定义，每个类和类的实例包装了属性和操纵这些数据的操作。最小的可测试单位是封装的类或对象，在面向对象的软件测试中，单元测试其实是对类的测试。

7.1.1 单元测试

单元测试是开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常，一个单元测试是用于判断某个特定条件（或者场景）下某个特定函数的行为。其中单元测试的方法主要有以下几类：

（1）人工静态分析：通过人工阅读代码的方式来查找代码中存在的错误。

（2）人工动态测试：人工设定程序的输入和预期输出，执行程序判断实际输出是否符合预期，若不符则自动报告错误。

（3）自动静态分析：使用代码复查工具，主要用来发现语法特征错误。

（4）自动动态测试：用工具自动生成测试用例并执行被测程序，用来发现行为特征错误。

7.1.2 类测试

面向对象的单元测试其实是对类的测试，类测试是在软件开发过程中进行的最低级别测试活动，它是可重复的。冯莉[28]给出了类测试的几种方法如下：

1. 基于状态的类测试

执行某个方法或序列后检测被测对象，把被测对象看作一个有限状态机，判断它是否会转移到预期状态的过程。这里的“状态”是指设计状态和实现状态：设计状态是指建模阶段认定的状态，处于较高的抽象层次，不依赖于实现时的具体表示；实现状态是指在实现层次上由对象的属性值确定的状态。

2. 基于规范的类测试

以需求和功能规范为基础的测试。通过分析软件的需求和功能来选择和产生测试数据，重点测试一个被测类的对象的消息序列是否将该对象置于“正确”的状态。

3. 基于 UML 的类测试

UML 是一种图形化的设计语言，UML 中的类表述符号为矩形框，使用类标题、类属性和操作描述系统模型。在使用 UML 进行软件设计建模时，设计者必须根据所需要描述的对象和系统动作选择适当的 UML 图，以达到设计效果。

4. 基于数据流的类测试

基于数据流的类测试由传统的数据流测试发展而来，通过选择一定的测试数据，使程序按照一定变量的定义使用路径执行，并检查执行结果是否与预期相符，从而发现代码的错误。

7.1.3 类测试过程

类测试即包含设计又涉及到具体的编码，整个执行类测试过程比较复杂。类测试过程，大体可分为五步。

1. 构建单元级测试

构建单元级测试包含两个方面：

(1) 构建单元级测试的环境。

(2) 构建单元级测试驱动以支持自动化测试。

2. CRC(Class Responsibility Collaborator)卡

CRC 是一种面向对象的分析技术，在一个面向对象开发项目中包括用户、分析员和实现者。在建模和设计过程中经常应用 CRC 卡(cards)，便于开发团队操作，主要由类、职责、协作三部分构成。

3. 测试驱动

测试驱动通过面向对象的方法、消息类、继承、封装、多态和实例等机制构造了整个软件系统的测试驱动程序，并对代码复用提供了保障。

4. 编码重构

在不改变代码外部行为的情况下对代码进行修改，首先编写测试代码验证业务代码，测试通过后，对业务代码进行更好的设计优化，使用测试代码回归验证设计优化是否有效正确。

5. 测试集成自动化

通过对读取的测试数据和测试驱动程序进行自动化操作，操作完成后对执行的结果进行信息化处理，生成了所有集成化的信息报表。

7.1.4 测试用例应用

软件公司追求的目标是以最少的人力、资源投入，在最短的时间内完成测试，保证软件的优良品质。测试用例是测试工作的指导，是软件测试必须遵守的准则，更是软件测试质量稳定的根本保障。

测试用例(test case)是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果的条件或变量，以便测试某个程序路径或核实是否满足某个特定需求，将测试行为具体量化，测试用例内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等并形成文档。

为了节省时间和资源、提高测试效率，必须从数量极大的可用测试数据中挑选出具有代表性或特殊性的测试数据来进行测试。目前研究测试过程，所有的测试用例都放在测试大纲中，使用测试大纲的好处：

- 保证测试功能不被遗漏。
- 功能不被重复测试，合理安排测试人员。
- 软件测试不依赖于个人。

1. 测试用例内容

实施一次测试而向被测系统提供输入数据、操作或各种环境设置。测试用例的设计与生成是依据测试大纲对其中每个测试项目进一步实例化。比如，对于一个输入项的测试，应当设计一组测试数据，包括合法的、边界的和非法的数据等。

2. 测试用例设计生成的基本准则

测试用例的代表性：能够代表并覆盖各种合理和不合理、合法和非法、边界和越界以及环境设置等。

测试结果的可判定性：测试执行结果的正确性是可判定，每一个测试用例都应有相应的期望结果。

测试结果的可再现性：相同的测试用例，系统的执行结果应当是相同的。

3. 确定测试用例

根据已列出的等价类表，按以下步骤确定测试用例：

1）每个等价类规定一个唯一的编号。

2）设计一个新的测试用例，尽可能多地覆盖尚未覆盖的有效等价类。重复这一步，所有有效等价类均被测试用例所覆盖。

3）设计一个新的测试用例，使其只覆盖一个无效等价类。重复这一步所有无效等价

类均被覆盖。

4. 编制测试用例

1）测试用例文档

编写测试用例文档应有文档模板，须符合内部的规范要求。测试用例文档将受制于测试用例管理软件的约束。软件产品或软件开发项目的测试用例一般以该产品的软件模块或子系统为单位，形成一个测试用例文档。

测试用例文档由简介和测试用例两部分组成。简介部分编制了测试目的、测试范围、定义术语、参考文档、概述等。每个具体测试用例包括下列详细信息：用例编号、用例名称、测试等级、入口准则、验证步骤、期望结果、出口准则、注释等。

2）测试用例设置

按功能、路径混合模式设置测试用例。按功能测试是最简捷的，按用例规约遍历测试每一功能。按路径混合测试优点在于可以避免漏测试，但存在局限性，如果一个子系统有很多模块，模块之间相互联系，路径数量成几何增长，路径数目太多很难测试。

3）设计测试用例

测试用例可以分为基本事件、备选事件和异常事件。

设计基本事件的用例，参照设计规格说明书，依据关联的功能、按路径分析法设计测试用例。

设计备选事件和异常事件的用例，需要在设计阶段形成的文档对备选事件和异常事件分析描述，而测试本身则要求验证全部非基本事件，并尽量发现其中的软件缺陷。

7.1.5 测试驱动

类测试驱动器是可执行程序，可以执行一个或者多个测试用例，王东刚[29]给出类测试驱动的三种方法，分别是调用 Main 方法、嵌入静态测试方法和实现独立测试类。通常对一个需要测试的类或接口提供一个测试类作为驱动(driver)，而根据类所需要调用的模块或对象提供一个或多个桩(stub)或对象。驱动的测试类需要根据设计说明书的要求，定义测试类的规格，形成测试用例，根据测试用例逐个调用被测试类的接口。

测试驱动程序是一个运行测试用例并收集运行结果的程序。测试驱动程序的设计相对简单，且易于适应它所测试类的增量说明的变化。图 7-1 为测试驱动程序框架图。

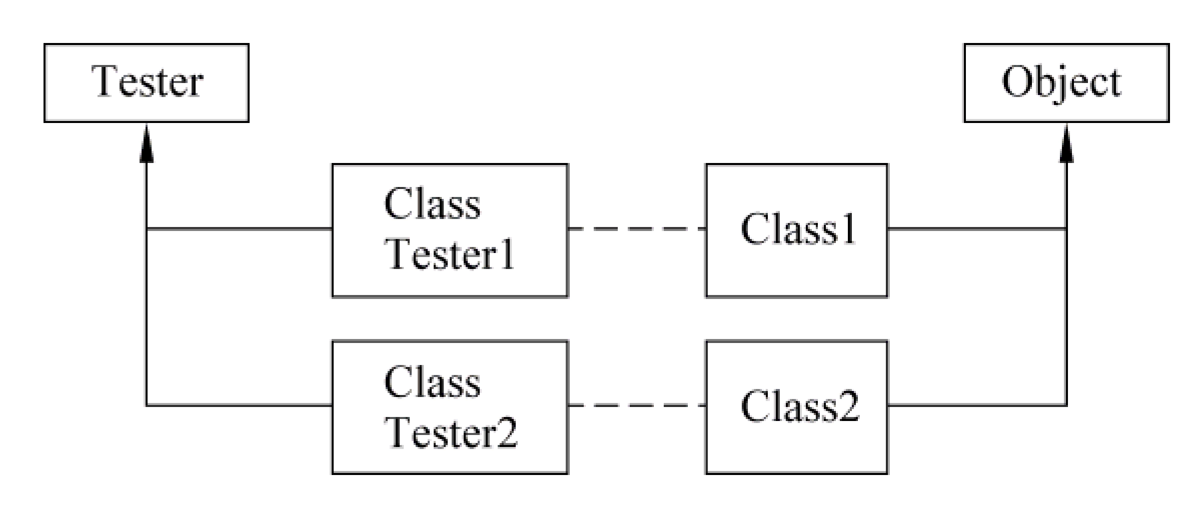


图 7-1 测试驱动程序框架图

测试驱动开发的主要思想是测试先行，在写一个类的具体实现之前，先写类的测试

代码,类的测试代码通过调用各种 public 方法对类进行测试。在写测试代码的过程中,思考类的调用方法,考虑类的封装。通过编写信任的测试,只要测试全部通过,那么说明类的方法全部正确。通过编写一个调用 yunsuan 类代码样例来说明,该段代码实现简单的加、减运算:

```
public class testyunsuan extends TestCase{
    public void testSum(){
        yunsuan suan=new yunsuan;
        assertEquals("6加 3等于 9",9,suan.sum(6,3));
    }
}
    public class yunsuan {
    public int sum(int x,int y){
        return 9;
    }
}
```

运行测试,测试通过。但是因为测试不够全面,重新修改测试:

```
public class testyunsun extends TestCase{
    public void testSum(){
        yunsuan suan=new yunsuan ;
        assertEquals("1加 2等于 3",3,suan.sum(1,2));
        assertEquals("2加 2等于 4",4,suan.sum(2,2));
            }
}
```

然后对 yunsuan 类的 sum 方法进行重构:

```
public int sum(int x,int y){
    return x+y;
}
```

在实际的开发中,依据上面的代码严格说还需要考虑临界值的测试,代码设计过程中重构是为了保证代码功能上的正确性,提高了代码的质量。

7.1.6 单元测试扩展

测试构建的延伸主要是针对类测试构造思想、接口类、抽象类等测试驱动程序的实现。父类中被测试用例所测试的代码被子类继承,类的创建自上而下,构建测试类同样采用自上而下的方法,便于实现。以下给出了简单的类测试构造方法,如图 7-2 所示。

dot_1 类有两个实例方法 name1 和 name2,dot_2 类继承 dot_1 类并且实现了新的实例 name3,dot_3 类继承了 dot_2 类,覆盖了 dot_2 类的实例 name2 和实例 name3,通过继承类之间关系来

dot_1
name1() name2()

dot_2
name3()

dot_3
name2() name3()

图 7-2 类的测试构造

确定继承的测试用例中是否需要产生新的子类测试用例，图 7-2 中测试用例在子类中的执行方式[29]，如表 7-1 所示。

表 7-1 构建测试类

类	继承类	类方法	是否改变	是否增加测试用例
dot_1		name1()		
		name2()		
dot_2	dot_1	name1()	否	否
		name2()	否	否
		name3()	是	是
dot_3	dot_2	name1()	否	否
		name2()	是	是
		name3()	是	是

根据表 7-1 可以推出测试用例拓展原则，如果子类添加了一个或者多个新操作，或者子类方法覆盖父类方法且重写了该方法，则需增加相对应的测试用例。

7.2 JUnit 骨架

JUnit 是用来做单元测试的一个工具，是由 Erich Gamma 和 Kent Beck 编写的一个回归测试框架。JUnit 测试是程序员测试，即所谓白盒测试，因为程序员知道被测试的软件如何(How)完成功能和完成什么样(What)的功能。JUnit 本质上是一套框架，即开发者制定了一套条框，遵循该条框按要求编写测试代码，如继承某个类，实现某个接口，就可以用 JUnit 进行自动测试了。

JUnit 是一款开源软件，支持语言 Smalltalk、Java、C++、Perl 等，支持的 IDE 为 JBuilder，VisualAge、Eclipse 等。

7.2.1 JUnit 设计原则

在使用 JUnit 时可以使测试代码与产品代码分开。针对某一个类的测试代码通过较少的改动便可以应用于另一个类的测试。由于 JUnit 是公开源代码的，可以进行二次开发。可以方便地对 JUnit 进行扩展。

JUnit 所派生的测试代码是为了验证被测试代码的实现是否符合预定设计而存在，测试编写原则如下：

(1) 简化测试的编写，这种简化包括测试框架的学习和实际测试单元的编写。

(2) 测试单元保持持久性。

(3) 利用既有的测试来编写相关的测试。

7.2.2 JUnit 安装

JUnit 是 Java 社区中知名度最高的单元测试工具。它诞生于 1997 年，由 Erich Gamma 和 Kent Beck 共同开发完成。其中 Erich Gamma 是经典著作《设计模式：可复用面向对象软件的基础》一书的作者之一，并在 Eclipse 中有很大的贡献；Kent Beck 则是一位极限编程方面的专家和先驱。

http://www.junit.org 是 JUnit 的网站，从该网站可以下载最新版本的 JUnit 包，在站点上可以查询到 JUnit 相关资料，包括开发语言、最新更新日期、授权使用书等；在网站 http://java.exampleshow.com/sourcecode/Category:JUnit 可以寻找 JUnit 范例；网站 http://www.eclipse.org 是著名的 Java 整合开发环境，提供了对 JUnit 良好的支持。

下载最新版本的 JUnit4.9b3.zip，下载完以后解压缩到喜欢的目录下，假设是 JUNIT_HOME，然后将 JUNIT_HOME 下的 JUnit.jar 包加到系统的 CLASSPATH 环境变量中，对于 IDE 环境，需要用到的 JUnit 的项目增加到 Lib 中，其设置不同的 IDE 有不同的设置。

本节根据 JUnit 测试网站及其使用手册[30]和相关文献资料[31]，介绍 JUnit 的安装信息。

1. 独立 JUnit 测试环境的建立

1) JUnit 的安装

(1) 将 JUnit 压缩包解压到一个物理目录中(例如 D:\JUnit4.9b3)。

(2) 记录 JUnit.jar 文件所在目录名(例如 D:\JUnit4.9b3\JUnit-4.9b3.jar)。

(3) 进入操作系统(以 WindowsXP 操作系统为准)，选择“开始”→“设置”→“控制面板”。在控制面板选项中选择“系统”，单击“环境变量”，在“系统变量”的“变量”列表框中选择 CLASS_PATH 关键字(不区分大小写)，如果该关键字不存在则添加。

(4) 双击 CLASS_PATH 关键字添加字符串“D:\ JUnit4.9b3\ JUnit-4.9b3.jar”(注意，如果已经有别的字符串请在该字符串的字符结尾加上分号“;”)，确定修改后，JUnit 就可以在集成环境中应用了。

2) JUnit 的卸载步骤

(1) 将 JUnit 压缩包删除。

(2) 进入操作系统(以 WindowsXP 操作系统为准)，按照次序选择“开始”→“设置”→“控制面板”。在控制面板选项中选择“系统”，单击“环境变量”，在“系统变量”的“变量”列表框中选择 CLASS_PATH 关键字(不区分大小写)。

(3) 双击 CLASS_PATH 关键字，删除字符串 D:\ JUnit4.9b3\JUnit-4.9b3.jar。

3) 测试安装是否成功

(1) 批处理文本方式。运行命令 java JUnit.textui.TestRunner。

(2) AWT 图形测试运行方式。运行命令 java junit.awtui.TestRunner。

(3) 基于 Swing 图形测试运行方式。运行命令 java junit.awtui.TestRunner。

如果上述命令都运行无误，则安装成功。

2. Eclipse 开发环境建立 JUnit

Eclipse 是一个非常优秀的集成开发环境(Integration Development Environment IDE),由 IBM 开发,基于 Java 公开源代码软件。Eclipse 核心是动态发现插件体系结构,平台负责处理基本环境的后台工作,每个插件专注于执行少量的任务。通过集成大量的插件,Eclipse 的功能可以不断的扩展,支持各种不同的应用。集成了 CDT 的 Eclipse 提供了一个开发 C/C++ 程序的功能强大的 IDE,将一些烦琐的事物变得简单,提高项目的开发效率。安装 JDK Eclipse 是一个基于 Java 平台的开发环境,它本身也要运行在 Java 虚拟机上,还要使用 JDK 的编译器,因此必须首先安装 JDK,同时也是 Eclipse 运行的必须条件。

(1) 从 Sun 的官方站点 http://java. sun. com 下载 JDK Windows 版,然后运行 setup. exe 安装,可以自行设定安装目录,例如安装到 D:\software\j2sdk 目录下。接下来要配置环境变量,以便 Java 程序能找到已安装的 JDK 和其他配置信息。右键单击"我的电脑",选择"属性",在弹出的对话框中选择"高级",单击"环境变量"按钮, 继续单击系统变量下方的新建按钮,创建 JAVA_HOME 环境变量。系统变量创建完毕后,在系统变量列表中查找 Path 变量,单击"编辑",在最后添上 JDK 的可执行文件的所在目录,即%JAVA_HOME%\bin,对应目录便是 D:\software\j2sdk\bin,附加到 Path 中即可,注意要以分号";"隔开。

(2) 配置好 JDK 后,安装 Eclipse,从 Eclipse 的官方站点 http://www. eclipse. org 上下载 Eclipse SDK。下载后,将它们解压到同一个目录,无须安装,直接找到目录下的 eclipse. exe 运行,出现启动画面: 稍等片刻,Eclipse 界面就出来了。

(3) 配置 JUnit,选择菜单"文件"→"新建"→"项目",选择 Java 项目,单击 Next 按钮对项目命名保存后即可编写测试程序。

7.2.3 软件测试自动化骨架

JUnit 框架是一个典型 composite 模式,TestSuite 可以容纳任何派生自 test 的对象,当调用 TestSuite 对象 run()方法时,会遍历自己容纳的对象,逐个调用它们的 run()方法。先给出一些框架定义:

(1) TestCase: 测试用例,是对测试目标进行测试的方法与过程集合。

(2) TestSuite: 测试包,是测试用例的集合,可容纳多个测试用例。

(3) TestResult: 测试结果的描述与记录。

(4) TestListener: 测试过程中的事件监听者。

每个测试方法所发生与预期不一致的状况描述,称其测试失败元素(TestFailure)。其中 TestCase 是框架的核心,它建立了 TestCase 和 TestSuite 之间的关联,为整个框架做了扩展预留。其余的类用来支援 TestCase 类,TestSuite 用来聚合多个 TestCase,Assert 类实现期望值 expected 和实际值 actual 的验证,TestResult 收集所有测试用例执行后的结果。图 7-3 为 JUnit 自动化测试框架。

该测试框架采用了 composite 设计模式,TestSuit 可以容纳任何派生自 Test 的对象,一旦调用 TestSuit 对象的 run()方法,遍历它所容纳的对象,逐个调用 run()方法,最

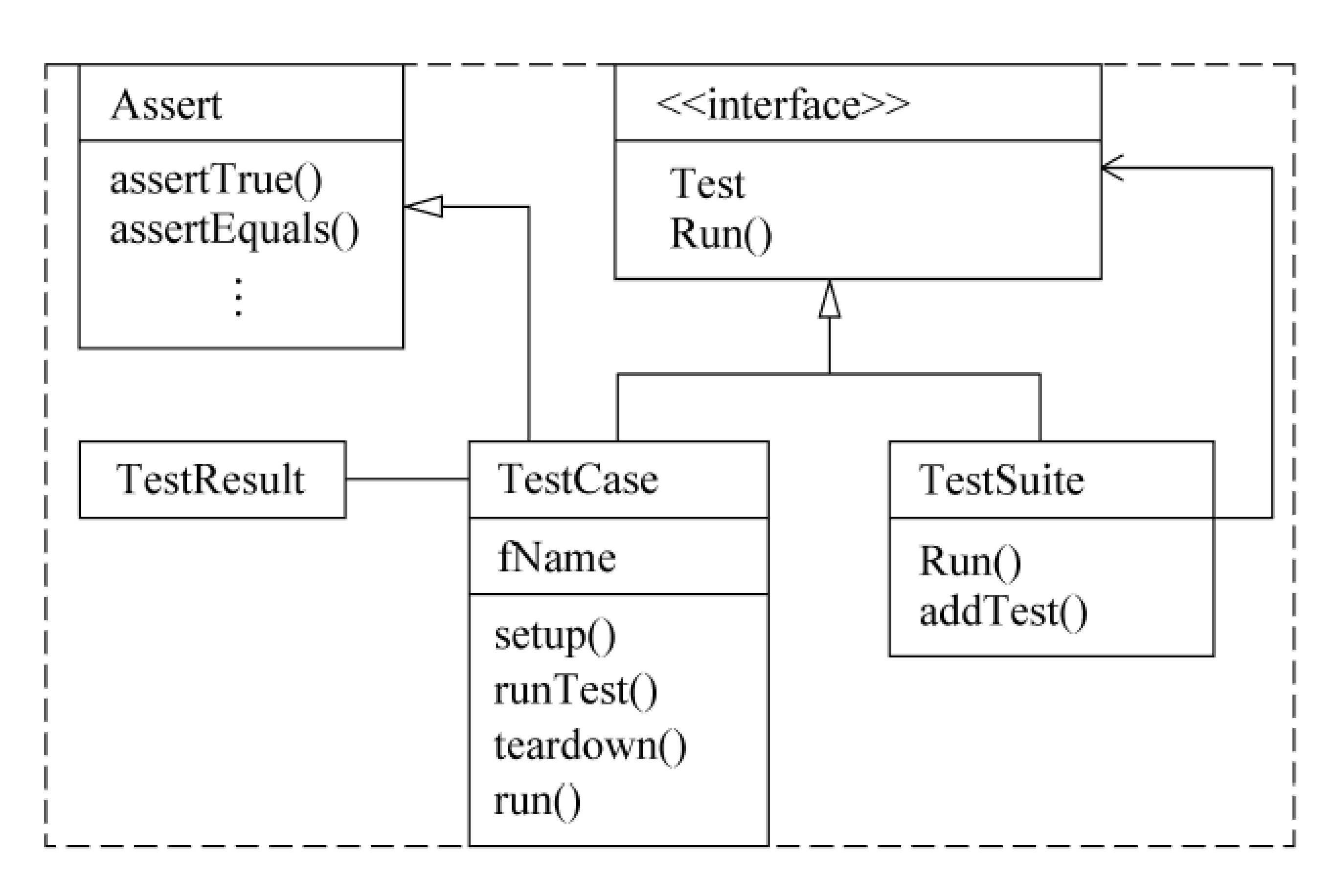


图 7-3　JUnit 自动化测试框架

后分析 run()方法返回的结果。

7.2.4　JUnit 断言

Assert 即 JUnit 断言，它包含了一组静态的测试方法，用于期望值和实际值比对是否正确。一旦即测试失败，Assert 类就会抛出一个 AssertionFailedError 异常，同时标志为未通过测试。如果该类方法中指定一个 String 类型的传参，则该参数将被作为 AssertionFailedError 异常的标识信息，告诉测试人员该异常的详细信息[29]。

Assert 方法封装了继承自 TestCase 最常见的测试任务，这些 Assert 方法可以极大地简化单元测试的编写，Assert 提供了 8 个核心方法如表 7-2 所示。

表 7-2　Assert 方法

assertTrue	断言条件为真。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertFalse	断言条件为假。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertEquals	断言两个对象相等。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertNotNull	断言对象不为 Null。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertNull	断言对象为 Null。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertSame	断言两个引用指向同一个对象。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
assertNotSame	断言两个引用指向不同的对象。若不满足，方法抛出带有相应的信息（如果有的话）的 AssertionFailedError 异常
Fail	让测试失败，并给出指定的信息

JUnit 提供了 6 大类 31 组断言方法，包括基础断言、数字断言、字符断言、布尔断言、比特断言、对象断言。王东刚[29]给出各种断言的归类讨论，如表 7-3 所示。

表 7-3　断言方法列表

序号	作用域	方 法 名 称
1	基础断言(Base)	assertTrue(String message,boolean condition)
		assertTrue(boolean condition)
		assertFalse(Sring message,boolean condition)
		assertFalse(boolean condition)
2	数值断言(Number)	assertEquals(String message,double expected,double actual,double delta)
		assertEquals(double expected,double actual,double delta)
		assertEquals(String message,float expected,float actual,double delta)
		assertEquals(float expected,float actual,float delta)
		assertEquals(String message,long expected,long actual)
		assertEquals(String message,short expected,short actual)
		assertEquals(Short expected,short actual)
		assertEquals(String message,int exoected,int actual)
		assertEquals(int expected,int actual)
3	字符断言(String)	assertEquals(String message, String expected,double actual,double delta)
		assertEquals(String expected,String actual)
		assertEquals(String message,char expected, char actual)
		assertEquals(char expected,char actual)
4	布尔断言(Boolean)	assertEquals(String message,Boolean expected,Boolean actual)
		assertEquals(Boolean expected,Boolean actual)
5	比特断言(Byte)	assertEquals(String message,Byte expected,Byte actual)
		assertEquals(Byte expected,Byte actual)
6	对象断言(Object)	assertEquals (String message,Object expected, Object actual)
		assertEquals(Object expected,Object actual)
		assertNotNull(String message,Object object)
		assertNotNull(Object object)
		assertNull(String message, Object object)
		assertNull(Object object)
		assertSame(String message, Object expected, Object actual)
		assertSame(Object expected, Object actual)
		assertNotSame(String message, Object expected, Object actual)
		assertNotSame(Object expected, Object actual)

关于 Assert 类的断言使用通过检查被测试的代码，判断是否返回了正确的结果，举一个 work 代码实例[29]如下：

```
Public void testwork(){
//创建一个工作实例,根据 work类的静态方法设置 work类型返回
work a= Greatwork.setworkStyle(new work("A"));
//由于 a实例是由方法生成,需要测试该方法是否返回了正确的 Null
assertNotNull("Null a object",a);
//测试该工作的准备工作是否就绪
assertTrue(a.is OK());
//测试工作的名称是否和设定的名称一致
AssertEquals("name check","a",a.getwork());
try{
//故意传入错误的对象测试 Greatwork实例是否能找出异常
Greatwork.setworkStyle(new work(Null));
Fail("gotten a workNotFoundException");
}
Catch(PlaneNotFouneException)
//找到了异常的信息处理
}
```

7.2.5 理解测试用例

TestCase 即测试用例，用抽象类来定义测试中的固定方法。TestCase 是 Test 接口的抽象实现。

其中，Test 接口是运行测试和收集测试的结果，它使用了 Composite 设计模式，Test 接口定义了两个方法：public int countTestCase()和 public void run(TestResult)。

public int countTestCase()：统计执行的测试用例(TestCase)的数量。如果是测试包则返回整个包中所有测试用例的统计值。

public void run(TestResult)：该方法有传递参数 TestResult，test 接口强制每个测试的执行都必须嵌入一个 TestResult 对象，该对象用来启动测试，记录测试过程。

考虑到 TestCase 是一个抽象类，只能被继承，不能被实例化。其构造函数可以根据输入的测试名称来创建一个测试用例，当启动 TestCase 的 run()方法后实时创建一个 TestResult 实例，然后按照一定的顺序运行整个测试过程。提供测试名的目的在于方便测试失败是查找失败的测试用例。编写 TestCase 的子类用于测试时，需要注意以下事项。

(1) 一次测试只测试一个对象，这样容易定位出错的位置。对于一个 TestCase，只测试一个对象，一个测试方法只测试一个对象中的方法。

(2) 最好在 Assert 函数中给出失败的原因，这样有助于查错。

(3) 在 setUp 和 tearDown 中的代码不应该与测试有关，而应与全局相关。

采用 TestCase 进行测试时，提供了 10 种测试方法如表 7-4 所示。

表 7-4　TestCase 测试方法

方　法	描　述
CoutTestCases	计算 run(TestResult result)所执行的 TestCase 的数目(由 Test 接口规定)
CreateResult	创建默认的 TesrResult 对象
getName	获得 TestCase 的名字
run	运行 TestCase,并收集 TestResult 中的结果(有 Test 接口规定)
runBare	运行测试序列,但不执行任何特殊功能,比如发现 Test 方法
setName	设置 TestCase 的名字
Setup	初始化 fixTrue,例如打开网络连接,这个方法会在测试执行之前被调用(由 Test 接口规定)
Teardown	销毁 fixTrue,例如关闭网络连接,这个方法会在测试执行之后被调用(由 Test 接口规定)
ToSteing	返回 TestCase 的字符串表示
runTest	重载并运行测试并断言其状态

创建一个简单的被测试类 public class exp，通过 public string change()返回一个给定的字符串 change(),通过继承 TestCase 抽象类构造自己的测试类,样例代码[29]如下。

```
public class exp_TestCase extends TestCase{
private exp exp_a;
//继承父类构造方法
  public exp_TestCase(){
  super();
}
protected void setup(){
  exp_a=new exp();
}
protected void tearDown(){
  exp_a=Null;
}
//创建测试类方法 assertEquals 对字符串进行逻辑对比
public void testchange(){
assertEquals("testchange","change()123",exp_a.change());
}
//执行测试用例
public static void main(string[] args){
exp_TestCase cet=new exp_TestCase()
}
public void runtest(){
  testchange();
}
}
```

TestCase 定义测试方法并且运行整个测试,TestResult 监听整个测试过程并且为每个输入性测试用例定义一个异常监听者,同时运行 TestCase 测试运行序列。因为 Class_E_Testcase 匿名继承了 TestCase 的空构造方法传入了一个 NULL,用匿名内类覆盖 TestCase 的 runtest()方法。

分析一个 JUnit 的代码框架如下：

```
public class Testann extends TestCase{
    public Testann(String name){
        super(name);
    }
    //重写 setUp(). setUp()方法进行初始化工作
    public void setUp(){
        try {
            super.setUp();
        //初始化
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    //重写 tearDown(). tearDown()进行销毁工作
    public void tearDown(){
        try {
        //销毁代码
            super.tearDown();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    //测试方法
public void testann(){
}
}
```

代码继承了 TestCase 提供的类，通过带参数的构造方法，覆盖 setUp()、teardown()，而 void 不带参数的 testann()方法对类别里面的逻辑方法进行测试。

7.2.6 TestResult 类

TestResult 用来收集参数，它负责包装和运行所有的 TestCase 并收集 TestCase 执行结果。比如，使用 assert(60，result，0)如果结果不等于 60 则 JUnit 就会创建一个 TestFailure 对象，它会储存在 TestResult 当中。TestRunner 使用 TestResult 来报告测试结果。如果 TestResult 集合中没有 TestFailure 对象进度条就用绿色，否则进度条用红色并输出失败测试的数目和它们的 stack trace，将测试结果转发到 TestListener 处理。JUnit 共有两种异常形态："失败"和"错误"。失败：是可以预期的，代码的改变会造成断言失败，只要修正代码，断言就可以再次通过。TestResult 结果类和其他类与接口做如下归类：

(1) TestResult 结果类集合了任意测试累加结果，通过 TestResult 实例传递每个测试的 Run()方法。TestResult 在执行 TestCase 时如果失败会异常抛出。

(2) TestListener 接口是事件监听规约,可供 TestRunner 类使用。它通知 Listener 对象相关事件,方法包括测试开始 startTest(Test test),测试结束 endTest(Test test)错误,增加异常 addError(Test test, Throwable t)和增加失败 addFailure(Test test, AssertionFailedErrort)。

(3) TestFailure 失败类是个“失败”状况的收集类,解释每次测试执行过程中出现的异常情况。其 toString()方法返回“失败”状况的简要描述。

本节以一个 TestResult 为例,统计测试运行的数量。为了使用 TestResult,必须把它作为参数传给 TestCase.run()方法,并通知 TestResult 当前测试已经开始,TestResult 会跟踪统计运行了多少个测试,代码如下:

```
public void run(TestResult result) {
    result.startTest(this);                //通知 TestResult 测试开始
    setUp();
    runTest();
    tearDown();
}
public synchronized void startTest(Test test) {
    fRunTests++;
}
```

TestResult 中的 startTest()方法定义成同步,一个 TestResult 对象可以收集不同线程中的测试结果。通过运行 TestCase,收集 TestResult 中的结果,采用测试方法 run 构建一个无参数版本的 run()方法,创建自己的 TestResult 对象如下:

```
public TestResult run() {
TestResult result=createResult();
run(result);
return result;
}
protected TestResult createResult() {
return new TestResult();
}
```

如果测试一直正确地运行,可以不用写测试。如果了解测试故障,JUnit 在故障(Failure)和错误(Error)之间作了区分。故障是可预期的,用断言来检测,错误是不可预期的。故障标识为 AssertionFailedError 错误。为了从故障中区分不可预料的错误,故障用 catch 语句捕获,这样就保证了本测试之后的其他测试得以运行。

7.2.7 测试包的实现

TestSuite 是把多个相关测试归入一组便捷方式,若没有提供自己的 TestSuite,TestRunner 会自动创建一个默认的 TestSuite,不能满足时,可能会组合多个 suite,把它们作为主 suite 的一部分,这些 suite 来自几个不同的 package,TestSuite 在通常情况下

是用来组合测试的，如果同时执行多个TestCase需要借助TestSuite。TestRuner负责启动TestSuite，而在多个Test Case情况下，运行哪些TestCase由TestSuite来决定。

通常情况下TestAll类包括一个静态的suite()方法，这个方法会注册应用程序需要定期执行所有Test对象(包括TestCase对象和TestSuite对象)，下面是一个典型的TestAll类。

```
import junit.framework.Test;
import junit.framework.TestSuite;
import junitbook.sampling.TestDefaultController;
public class TestAll extend TestCase
{
    public static Test suite()
    {
        TestSuite suite= new TestSuite("All tests from exp");
        suite.addTestSuite(Testexp.class);
        suite.addTestSuite(TestDefaultController.class);
    return suite;
    }
}
```

由于TestSuite类组装多个TestCases，待测类中可能包含了对被测类的多个测试，而TestSuit负责收集这些测试。TestSuite实现了Test接口，且可以包含其他的TestSuites。它可以处理加入Test时所有抛出的异常。

TestSuite处理测试用例有6个规约[29](否则会被拒绝执行测试)：

A. 测试用例必须是公有类(Public)；

B. 测试用例必须继承于TestCase类；

C. 测试用例的测试方法必须是公有的(Public)；

D. 测试用例的测试方法必须被声明为Void；

E. 测试用例中测试方法的前置名词必须是Test；

F. 测试用例中测试方法无任何传递参数。

介绍TestSuite处理测试用例(TestCase)标准写法[29]如下：

```
//必须声明为 public class, 继承于 TestCase 类
public class Class_TestCase extends TestCase{
//必须声明 public
public class_TestCase(){
super();
}
//测试用例必须声明为 public,并且加上 test 修饰前缀
public void testAMethod(){
⋮
}
public void testBMethod(){
```

```
  ⋮
}
public void testCMethod(){
  ⋮
}
}
```

在使用 TestSuite 时通常需要定义一个 TestCase，并使用 TestRunner 来运行测试，事实上 TestRunner 并不直接运行 TestCase 上的单元方法，而是通过 TestSuite，TestSuite 可以将数个 TestCase 组装在一起，从而确保每个 TestCase 保持简单，举一个数学工具测试用例代码如下：

```
MathToolTest.java
package onlyfun.caterpillar.test;
import onlyfun.caterpillar.MathTool;
import junit.framework.TestCase;
public class MathToolTest extends TestCase {
    public MathToolTest(String testMethod) {
        super(testMethod);
            }
    public void testGcd() {
        assertEquals(2, MathTool.gcd(4, 2));
            }
    public static void main(String[] args) {
        junit.textui.TestRunner.run(MathToolTest.class);
            }
}
```

上面例子中没有看到任何的 TestSuite，TestRunner 会自己建立一个 TestSuite 并自动找出 TestCase()方法。如果自行生成 TestSuite，则在继承 TestCase 之后，提供静态的 suite()方法，例如：

```
public static Test suite() {
    return new TestSuite(MathTool.class);
}
```

如果没有提供任何的 TestSuite，则 TestRunner 会建立一个，并找出 TestCase()方法。如果要加入更多的测试方法，使用 addTest()，suite()方法传回一个 TestSuite，TestRunner 会调用 TestSuite 的 run()方法，然后 TestSuite 会将之委托给 TestCase 上的 run()方法，并执行每一个 TestCase()方法。

7.2.8 事件监听者实现

JUnit 提供了 TestListener 接口，以帮助对象访问 TestResult 并创建有用的报告。TestRunner 实现了 TestListener，可以有任意数量的 TestListener 向 JUnit 框架注册，

TestListener 可以根据 TestResult 提供的信息做需要做的任何事情。但是编写测试时不必实现这个接口,只有需要扩展 JUnit 框架时才会需要实现这个接口。

TestListener 接口抽象了所有测试监听者的行为,包括两个添加错误和失败的方法:addError (Test, Throwable) 和 addFailure (Test, AssertionFailedError),开始测试 startTest (Test)方法,结束测试 endTest(Test)方法。TestListener 是一个接口,定义几个方法,说明监听几个方法。如测试开始,发生失败,发生错误,测试结束等监听事件的时间点。由具体的类来实现。

```
//测试过程监听
public interface TestListener {
//错误发生
public void addError(Test test, Throwable t);
//失败发生
public void addFailure(Test test, AssertionFailedError t);
//测试开始
public void startTest(Test test);
//测试结束
public void endTest(Test test);
}
```

7.3 Eclipse 中 JUnit 的使用

Eclipse 自带 JUnit 插件,安装好 Eclipse 后不用安装 JUnit 便可以在项目中开始测试相关的类,并且可以调试测试用例和被测试类。关于 Eclipse 中 JUnit 安装查看 7.2.2 节内容。关于在 Eclipse 中 JUnit 的应用可以参考 JUnit 测试网站及其使用手册[30]和相关文献资料[31],操作过程如下:

打开 Eclipse 中的 JUnit,创建一个新的 Java Project,执行 File→New→Java Project,弹出图 7-4 所示 Java 工程对话框,输入 Project name 后,单击 Finish 按钮。

新建一个 Class,执行 File→New→Class,在 Java Class 对话框 Name 栏中输入 Class 名,选中 public static void main(string[] args),单击 Finish 完成 Class 的创建。

在 Java 工程界面中编辑自己设计的 Java 代码,编辑完成后保存,然后在 Eclipse 中加入一个 JUnit 库,在属性对话框中添加 Library,选择 JUnit,单击 Next 按钮选择 JUnit library version,单击 Finish 按钮完成 JUnit 库的添加。

新建一个 JUnit TestCase,执行路径 New→JUnit Test Case,输入 TestCase 名称后单击 Finish 按钮完成 TestCase 创建。

运行 JUnit Test,执行路径 Run As→JUnit Test,如果 Assert 成功,则会出现绿色的条带,如图 7-5 所示。

如果有任何一个方法没有断言(Assert)成功,就会显示红色的条带,如图 7-6 所示,并在下面的运行失败描述(Failure Trace)中说明原因。

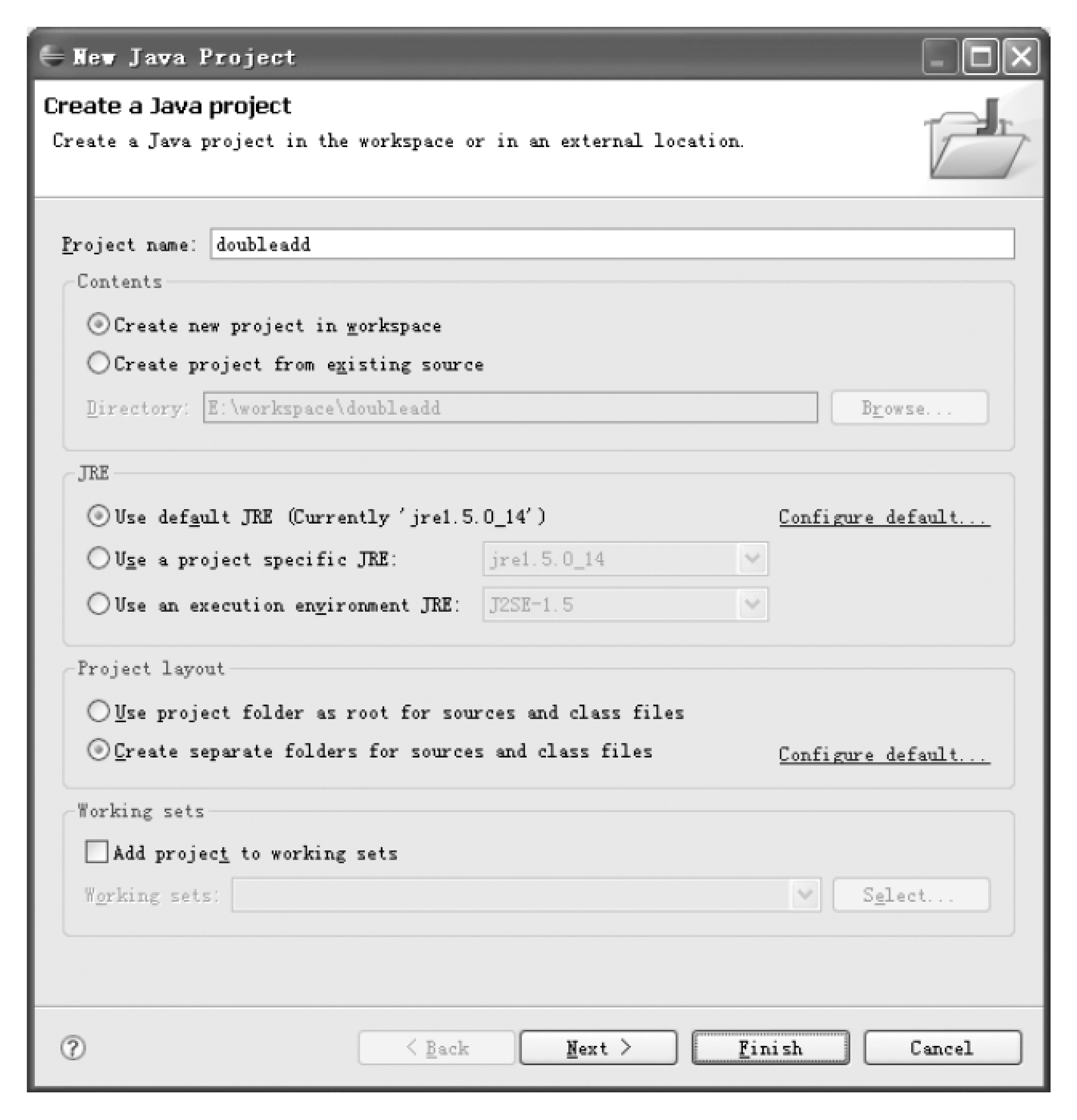


图 7-4　Java 工程对话框



图 7-5　运行正确

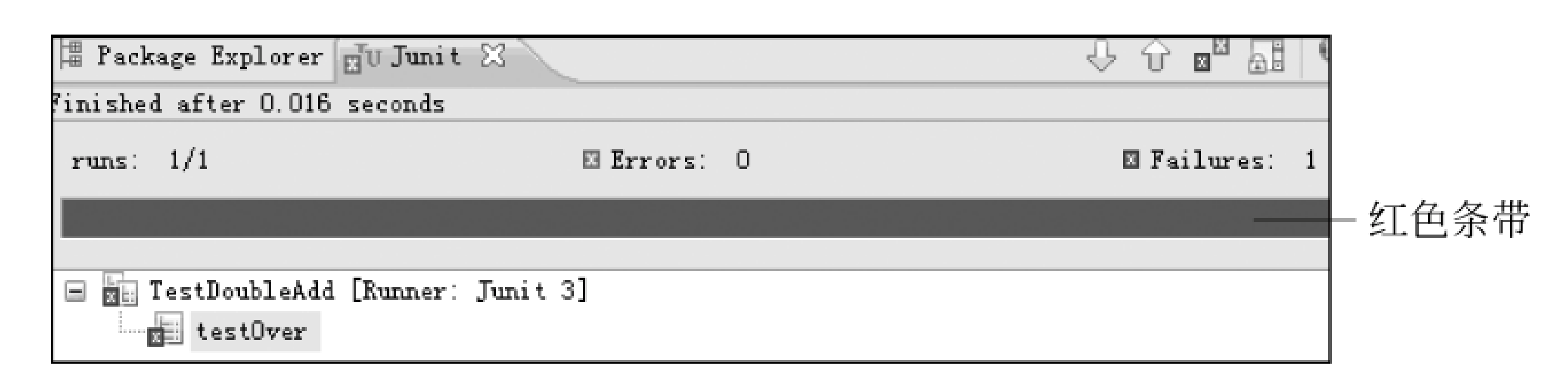


图 7-6　运行错误

以上简单介绍了在 Eclipse 中 JUnit 的快速应用，方便初学者快速在 Eclipse 环境下进行 JUnit 操作。

第 8 章



负载测试

负载测试是系统在资源超负荷情况下的表现，而测试对象承担相应的工作任务，在不同工作环境下评估系统的性能，目的是保障系统在超负荷情况下仍能正常工作。而性能的负载测试是软件测试的重中之重，测试工具 LoadRunner 通过模拟用户实时监测系统以确认和查找问题，并对企业架构进行测试。LoadRunner 是一种适用于各种体系架构的自动负载测试工具，它以其界面友好、方便易用、功能强大等特点获得广泛的使用。

8.1 LoadRunner 程序安装

LoadRunner 可以安装在 Windows 系统或者 Linux 系统下，本节只介绍 LoadRunner 在 Windows 系统下安装的过程。

8.1.1 Windows 系统下 LoadRunner 的安装

运行压缩的安装程序文件 HP Mercury LoadRunner V8.0.exe，则出现图 8-1 Installshield Wizard 对话框，选择好安装路径，单击 Next 按钮。注意，不要将 LoadRunner 安装在带有中文的路径下，否则在安装过程中可能会出现问题。连续单击 Next 按钮，根据提示进行安装，直到安装完成后选择立即重启或者以后启动，如图 8-2 所示，选择好选项后单击 Finish 按钮完成软件安装。

8.1.2 许可协议和样例安装

LoadRunner 安装完成后，如果涉及 License 信息过期需要重新输入新的协议，可以通过单击启动界面的 License 工具栏，在 LoadRunner License Information 对话框中单击 New License 按钮，弹出 License 对话框输入新的许可协议，单击 OK 按钮完成协议添加。

LoadRunner 本身自带了样例安装程序，单击“开始”→“程序”→Mercury→Samples Setup，出现如图 8-3 所示界面。单击 Next 按钮，选择要安装的组件，连续单击 Next 按钮，组件安装完成，如图 8-4 所示。

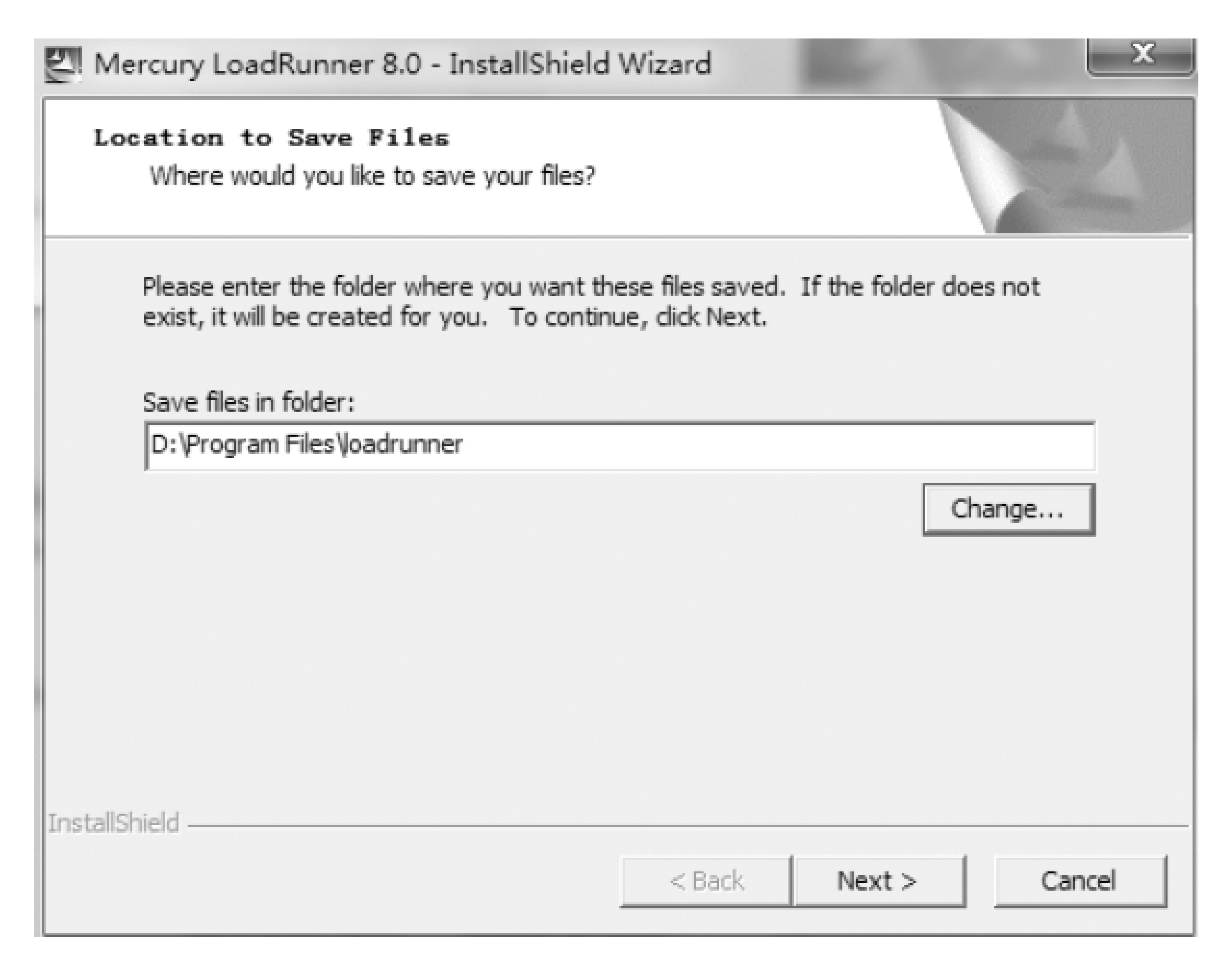


图 8-1 Installshield Wizard 对话框

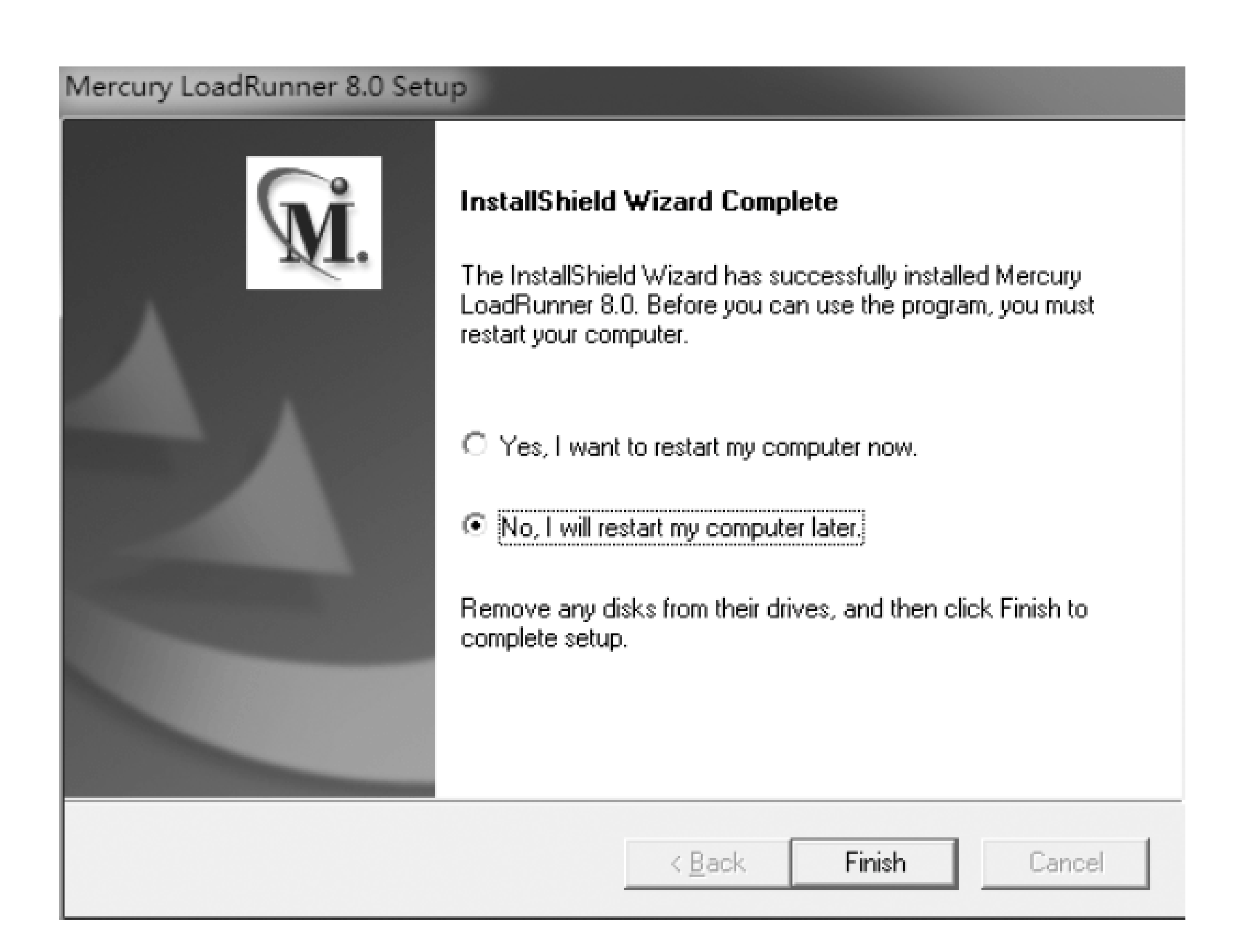


图 8-2 安装完成界面

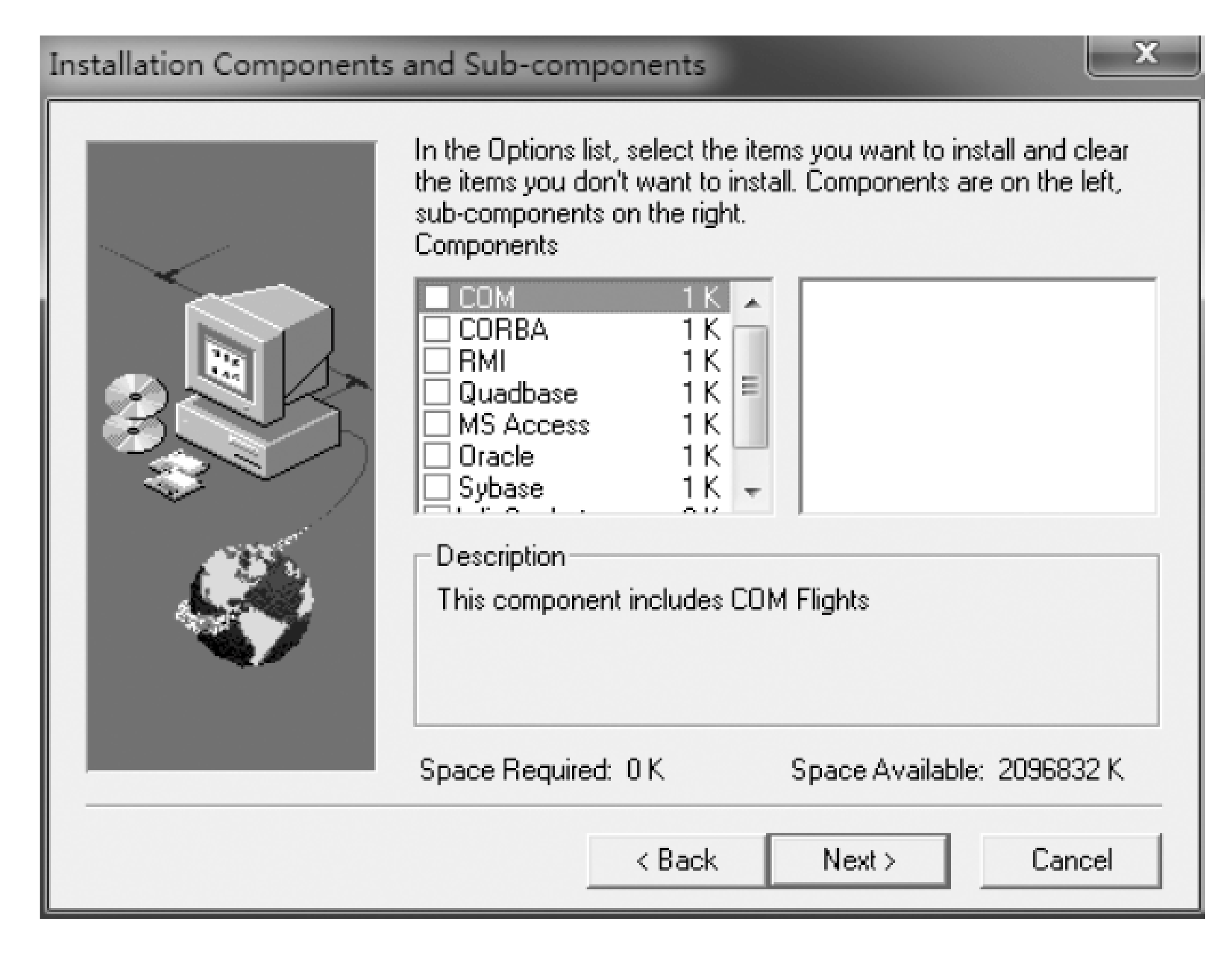


图 8-3 组件选择界面

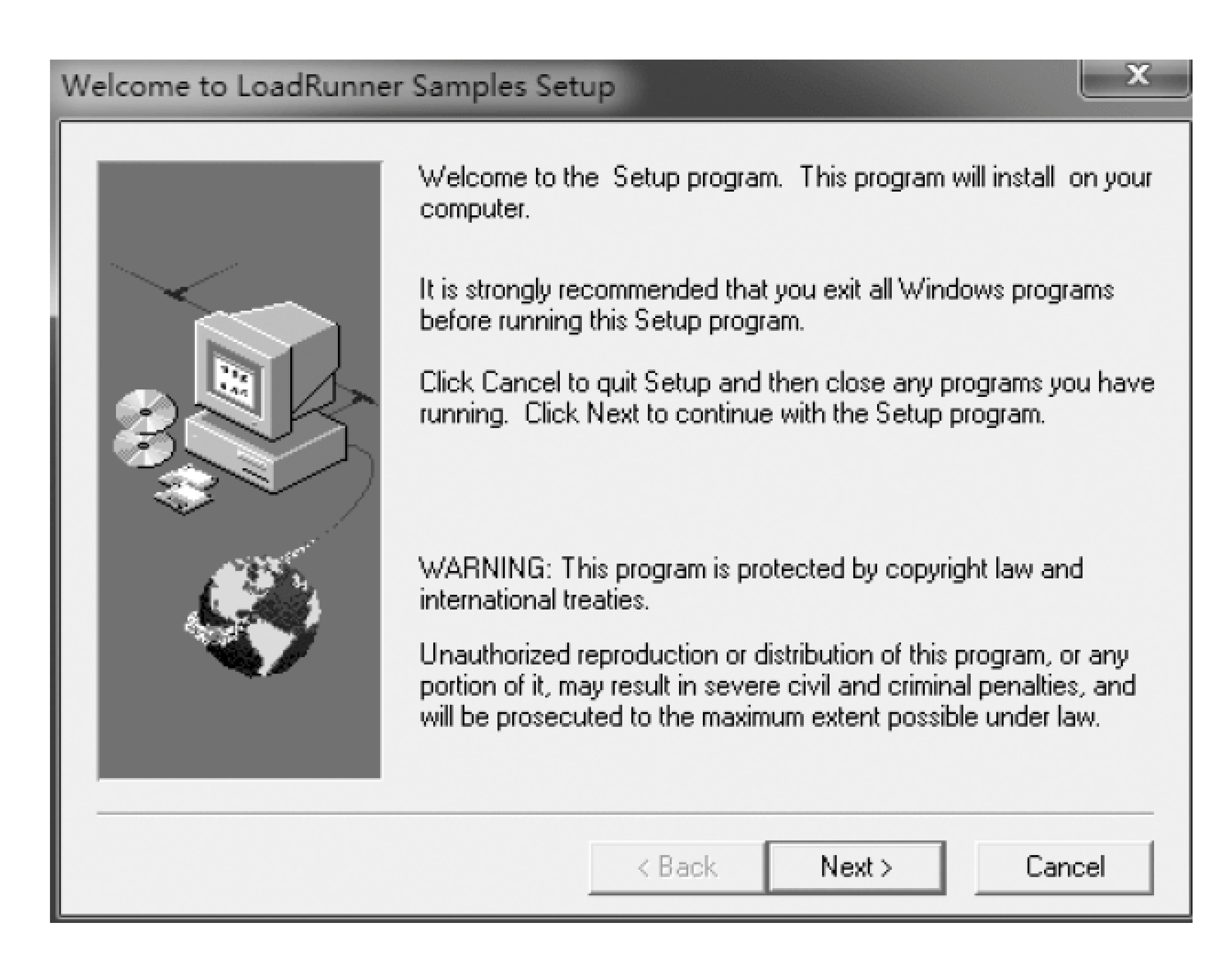


图 8-4 组件安装完成界面

8.2 LoadRunner 简介

目前,随着软件开发技术快速发展,现代应用软件的复杂性也在急剧上升,软件用户关注的内容不仅体现在功能实现的正确性,其系统的性能表现也是用户关注的重点,其中性能测试是测试系统的主要手段,是软件测试的重点。使用 Mercury LoadRunner 进行自动负载测试是应用程序部署过程中必不可少的部分。通过 LoadRunner 的使用不仅可以轻松创建虚拟用户和真实的负载,而且对系统的性能进行实时监测并精确分析测试结果。

1. LoadRunner 关键组件

(1) 虚拟用户生成器:捕获最终用户业务流程并创建自动性能测试脚本。

(2) Controller:组织、驱动、管理和监控负载测试。

(3) 负载生成器:通过运行虚拟用户生成负载。

(4) Analysis:查看、分析和比较性能结果。

2. LoadRunner 关键术语

(1) 场景:根据性能需求在每一个测试运行期间定义发生的事件。

(2) 集合点:同步虚拟用户以便在同一时刻执行任务,集合点可以实现用户同步。

(3) 事务:度量最终用户业务流程。一个完整的事务由事情开始、事务结束以及一个或多个业务操作构成。

(4) VuGen:模拟实际用户的操作来使用应用程序。

(5) 思考时间:用户执行两个连续操作等待的时间。

3. 负载测试流程

（1）创建脚本：捕获应用程序中执行的最终用户操作。
（2）设计场景：设置负载测试场景。
（3）运行场景：运行并监控负载测试。
（4）分析结果：分析负载测试期间 LoadRunner 生成的性能数据。

8.3 协议选择

LoadRunner 支持两项最广泛的协议：WAP 和 I-mode，通过负载测试系统整体架构，允许只通过记录一次脚本，即可检测到无线互联网系统。LoadRunner 的虚拟用户生成器本身提供了多种协议，方便用户模拟系统的 Vuser 技术。各种 Vuser 技术既可以使用单协议，又可以使用多协议。协议选择时需要根据字母顺序查看所有支持的协议列表。

应用 LoadRunner 进行性能测试时，有时在脚本录制完成后，脚本编辑框中没有产生任何脚本代码。例如，一个 FTP 的应用程序，选择用“(HTTP/HTML)”协议录制，那么录制完成后，就会产生一个空的脚本，主要原因是选择了错误的协议。使用 LoadRunner 时要正确选择协议，这关系到脚本能否正确录制与执行，因此在进行程序的性能测试之前，测试人员必须弄清楚被测试程序使用的是什么协议。下面是记录脚本完成网络检测的过程。

启动 Visual User Generator，通过菜单新建一个用户脚本，选择系统通信协议(见图 8-5)。文献[32]中给出了以创建一个 Web 类型的单协议脚本为例，确保“类别”类型为“所有协议”。VuGen 将显示所有可用于单协议脚本的协议列表。向下滚动该列表，选择“Web (HTTP/HTML)”并单击“确定”按钮，创建一个空白 Web 脚本。Web(HTTP/HTML)协议确定后，进入主窗体，通过菜单 Vuser 选择启动录制脚本命令。



图 8-5 协议选择界面

8.4 创建脚本

LoadRunner 的 VuGen 利用虚拟用户可以同时产生成千上万个用户访问，通过建立系统负载生成虚拟用户来模拟真实用户的业务操作，通过记录业务流程将其转化为测试脚本。脚本建立后，进行参数化操作，利用实际发生数据来测试应用程序，从而反映出系统的负载能力，所以 LoadRunner 极大地减少了负载测试所需的硬件和人力资源。

8.4.1 虚拟用户生成器

LoadRunner 虚拟用户生成器(VuGen)采用录制并播放机制。应用程序中根据业务流程操作时，VuGen 将这些操作录制到自动脚本中，以便作为负载测试的基础。

开始录制用户，打开 VuGen 并创建一个空白脚本。通过录制事件和添加手动增强内容来填充空白脚本，依据 MevcuryLoadRunner 8.1[33]，具体操作如下：

1. 启动 LoadRunner

选择"开始"→"程序"→ Mercury LoadRunner → LoadRunner。打开 Mercury LoadRunner Launcher，窗口如图 8-6 所示。

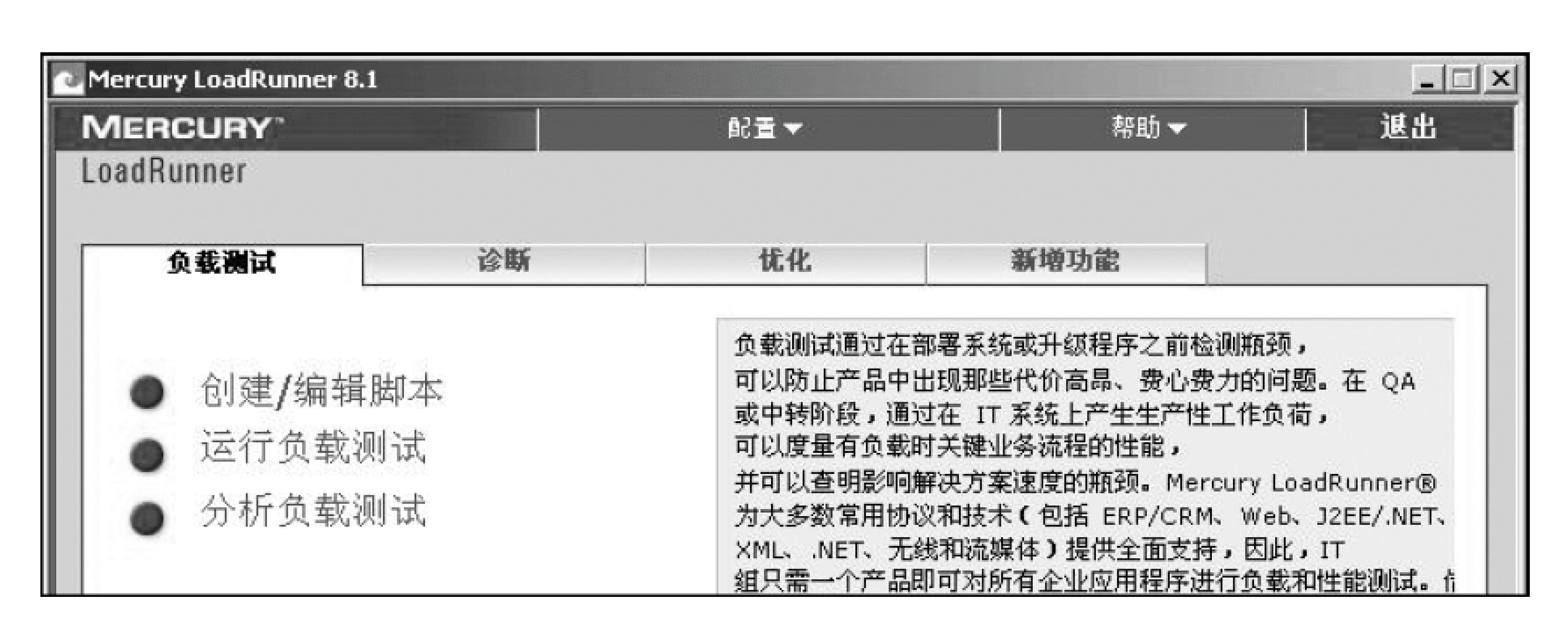


图 8-6 LoadRunner 启动窗口

2. 打开 VuGen

在启动窗口中，单击"负载测试"项目栏，单击"创建/编辑脚本"。打开 VuGen 的启动界面，如图 8-7 所示。

3. 创建空白 Web 脚本

在 VuGen 启动页的"脚本"选项卡中，单击"新建 Vuser 脚本"，打开"新建虚拟用户"对话框，如图 8-5 所示，其中显示了用于新建单协议的脚本选项。协议是客户端用来与系统后端进行通信的语言，选择好"Web(HTTP/HTML)协议"后单击"确定"按钮即成功创建空白 Web 脚本。



图 8-7　VuGen 启动界面

8.4.2　录制业务

8.4.1 节中,创建了一个空白 Web 脚本,将事件录制到脚本中。下面选取一位乘客预订从丹佛到洛杉矶的航班[33],以查看航班路线事件为例录制脚本操作。

1. 开始录制

单击任务窗格底部的“开始录制”按钮,如图 8-8 所示。也可以选择 Vuser→“开始录制”,或单击页面顶部工具栏中的“开始录制”按钮。

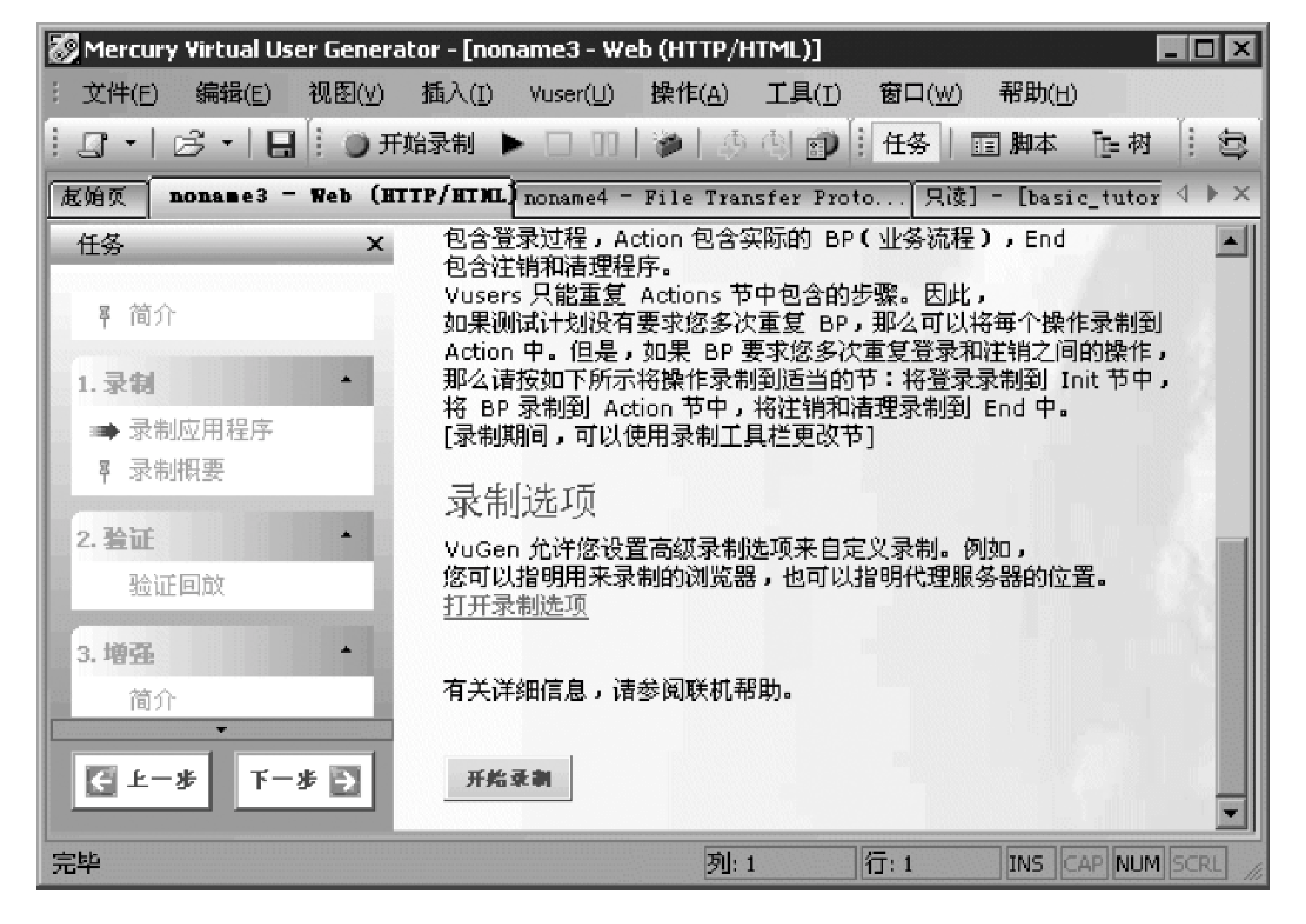


图 8-8　开始录制界面

在图 8-9URL 地址框中,输入 http://www.idc.ac.il。在“录制到操作”框中,选择 Action,单击“确定”按钮,打开一个新的 Web 浏览器,并显示 Mercury Tours 站点。启动服务器,选择“开始”→“程序”→Mercury LoadRunner→“示例”→Web→“启动 Web 服务器”。



图 8-9　录制对话框

2. Mercury Tours 网站的操作

在“成员姓名”框中输入 jojo，在“密码”框中输入 bean，单击“登录”，打开欢迎页面。单击“航班”，将打开“查找航班”对话框：

- 出发城市：丹佛(默认设置)；
- 出发日期：保持默认设置不变(当前日期)；
- 到达城市：洛杉矶；
- 返回日期：保持默认设置不变(第二天的日期)；
- 座位首选项：过道。

保持其余的默认设置不变，然后单击“下一步”，将打开“付费详细信息”对话框。输入付费信息并预订航班，在“信用卡”框中输入 12345678，在“输出日期”框中输入“06/06”。单击“下一步”，将打开“发票”对话框，并显示发票，其他操作可根据页面提示进行。

3. 停止录制过程

生成 Vuser 脚本时，选择“代码生成”后弹出如图 8-10 所示对话框。VuGen 向导将自动进入任务窗格中的下一步，并显示录制概要。如果没有看到概要，请单击任务窗格中的“录制概要”。

录制概要包括协议信息和会话执行期间创建的操作列表，对于录制期间执行的每个步骤，VuGen 都生成一个快照(即录制期间窗口的图片)。这些录制的快照的缩略图显示在右窗格中。如果需要重新录制脚本，请单击页面底部的“再次录制”按钮。

4. 文件保存

在“文件名”文本框中输入 basic_tutorial，并单击“保存”按钮。VuGen 将把该文件保存在 LoadRunner 脚本文件夹中，并在标题栏中显示该测试名称。

8.4.3　查看脚本

查看 VuGen 录制的脚本，可以在树视图或脚本视图中查看。

树视图是基于图标的视图。在树视图中查看脚本，选择“视图”→“树视图”或单击



图 8-10 “录制概要”对话框

“树视图”按钮。对于录制期间所执行的每一步骤，VuGen 都在测试树中生成一个图标和一个标题，在树视图中，将看到作为脚本步骤的用户操作。

脚本视图是一种基于文本的视图。在脚本视图中查看脚本，选择“视图”→“脚本视图”或单击“脚本视图”按钮。在脚本视图中，VuGen 将在编辑器中显示带有彩色编码的函数及其变量值的脚本。

8.5 编辑脚本

编辑脚本主要从插入事务（Transaction）、插入集合点（Rendezvous ）和插入检查点等多个方面进行处理。

1. 插入事务（Transaction）

为了衡量服务器的性能，需要在任务的开始和结束位置插入事务，每个事务度量服务器响应指定的 Vuser 请求所用的时间。事务可以是简单任务（如单个查询），也可以是复杂任务（如提交多个查询和生成报表）。在运行测试脚本时，LoadRunner 运行到某个事务的开始点时，LoadRunner 开始计时，直到运行到该事务的结束点，计时结束。

插入一个事务有两种方式实现，一种是手工方式；另外一种是利用菜单或者工具条进行事务的添加。要注意的是手工方式要求编写脚本人员必须十分清楚脚本的内容，在合适的位置插入事务的开始和事务的结束函数。选取菜单方式操作如下[34]：

（1）在 VuGen 界面选择 Insert→Start Transaction，弹出如图 8-11 所示对话框。

（2）输入该事务的名称。事务的名称最好能够说明该事务完成的动作。插入事务的

开始点后，需要在定义事务的操作后面插入事务的“结束点”，执行 Insert→End Transaction，弹出如图 8-12 所示对话框。



图 8-11 Start Transaction 对话框

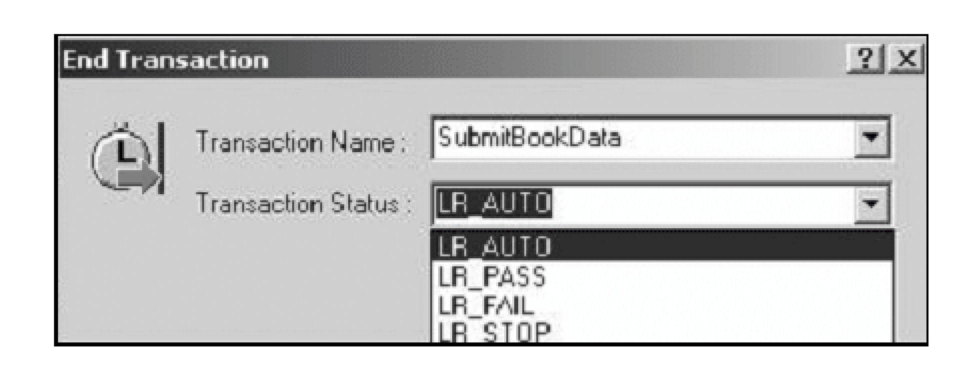


图 8-12 End Transaction 对话框

(3) 一般情况下，事务名称不用修改。事务的状态默认情况下是 LR_AUTO。在手工编写代码时，需要手动设置事务的状态。脚本中事务的代码如下：

```
Lr_start_transaction("submitbookdata");
//此部分代码是具体事务操作
Lr_end_transaction("submitbookdata",LR_AUTO);
```

2. 插入集合点(Rendezvous)

集合点是一个并发访问的点。在测试计划中，可能会要求系统能够承受 500 人同时提交数据，在提交数据操作前加入集合点。当虚拟用户运行到提交数据的集合点时，LoadRunner 就会检查同时有多少用户运行到集合点，如果不到 500 人，LoadRunner 会命令已经到集合点的用户在此等待，当在集合点等待的用户达到 500 人时，LoadRunner 就会命令 500 人同时去提交数据，达到并发访问的目的。操作如下：

(1) 在 VuGen 界面选择 Insert→Rendezvous，弹出如图 8-13 所示对话框。

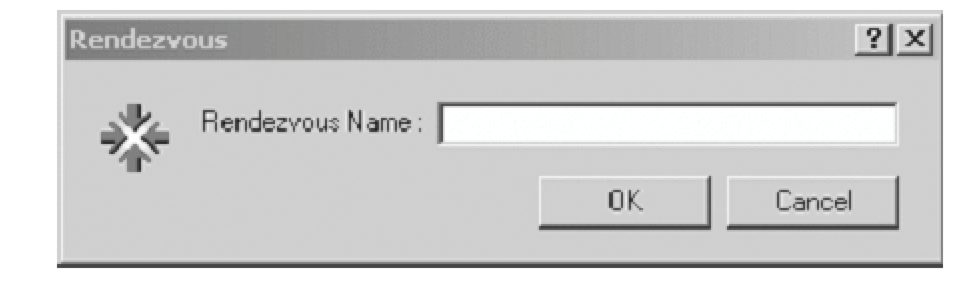


图 8-13 Rendezvous 对话框

(2) 输入该集合点的名称。

脚本中集合点的代码如下：

```
Lr_rendezvous("submitquerydata")
```

注意：

(1) 集合点经常和事务结合起来使用，常放在事务的前面。

(2) 集合点只能插入到 Action 部分，vuser_init 和 vuser_end 中不能插入集合点。

3. 插入检查点

在回放脚本期间，搜索特定的文本字符串或者图片等内容，从而验证服务器响应内容的正确性。例如验证一个用户是否成功登录到系统，通过设置一个文本或者图片检查点进行验证。插入一个检查点有两种实现方式，一种是手工方式；另一种方式利用菜单或者工具条。利用手工方式插入检查点时要求编写脚本人员清楚脚本内容，在合适的位置插入检查点函数。利用菜单方式时，操作步骤如下[34]：

(1) 切换到脚本 TreeView 视图，添加 Text 检查点，在录制过程中，也可以在录制完

成后添加。在树形菜单中选择需要插入检查点的一项，右击，如果检查点在该操作执行前，选择 Insert Before，否则选择 Insert After。然后在弹出的对话框中选择 Text Check（见图 8-14）。

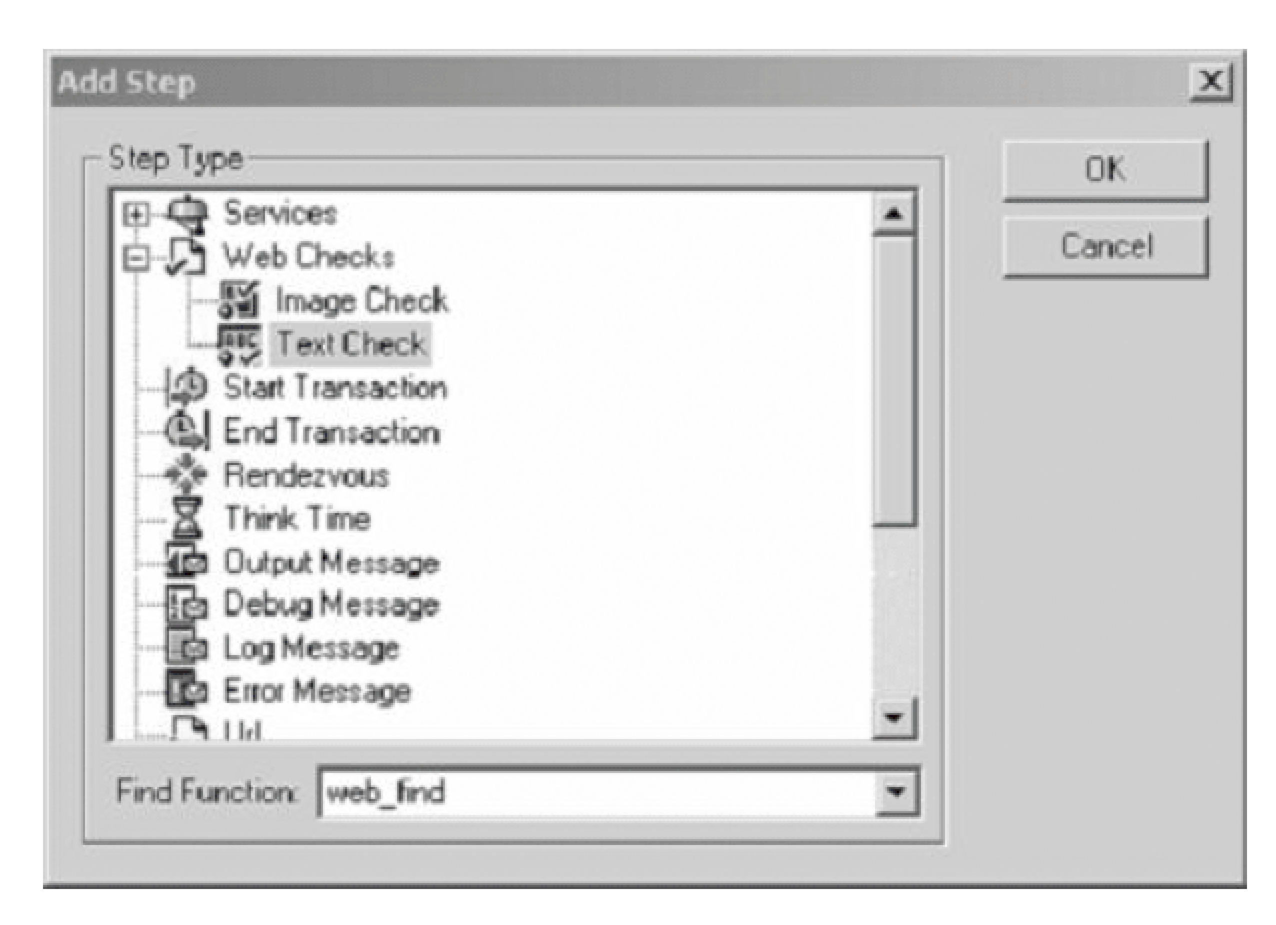


图 8-14 Add Step 对话框

（2）单击 OK 按钮后，出现 Text Check Properties 对话框，按照要求设置后，完成添加 Text 检查点。

8.6 负载测试与运行过程

Virtual Users 建立后，需要设定负载方案，业务流程组合和虚拟用户数量，LoadRunner 的 Controller 可以很快组织起多用户的测试方案。Controller 的 Rendezvous 功能提供一个互动的环境，既能建立起持续且循环的负载，又能管理和驱动负载测试方案。并且利用它的日程计划服务来定义用户在什么时候访问系统以产生负载，将测试过程自动化。

本节介绍测试负载的整个运行过程，并观察系统在负载下的行为。测试过程需要使用 LoadRunner Controller。

8.6.1 LoadRunner Controller 简介

LoadRunner Controller 提供所有需要的有助于创建并运行测试的工具，以准确地模拟工作环境。在本节中，目标是创建一个场景，通过模拟十个旅行代理同时登录系统、搜索航班、购买机票、查看路线和注销系统的行为[33]，创建新场景操作如下：

1. 打开 Mercury LoadRunner

选择“开始”→“程序”→ Mercury LoadRunner → LoadRunner。打开 Mercury LoadRunner Launcher 窗口。

2. 打开 Controller

在“负载测试”选项卡中，单击“运行负载测试”，打开 LoadRunner Controller。默认

情况下，Controller 打开时将显示“新建场景”对话框，如图 8-15 所示。

新建场景
选择场景类型
手动场景
通过指定要运行的虚拟用户数来管理负载测试
使用百分比模式在脚本间分配 Vuser
面向目标的场景
允许 LoadRunner 控制器基于指定的目标创建场景
选择要在场景中使用的脚本
可用脚本
场景中的脚本
添加(A) ==>>
删除(R)
浏览(B)...
录制(E)...
Quality Center...
启动时显示
确定
取消
帮助(H)

图 8-15　“新建场景”对话框

3. 选择场景类型

选择“手动场景”，通过手动场景，可以控制正在运行的 Vuser 数量及其运行的时间，还可以测试运行的 Vuser 数。

Controller 窗口的“设计”选项卡包含场景计划和场景组两部分，如图 8-16 所示。

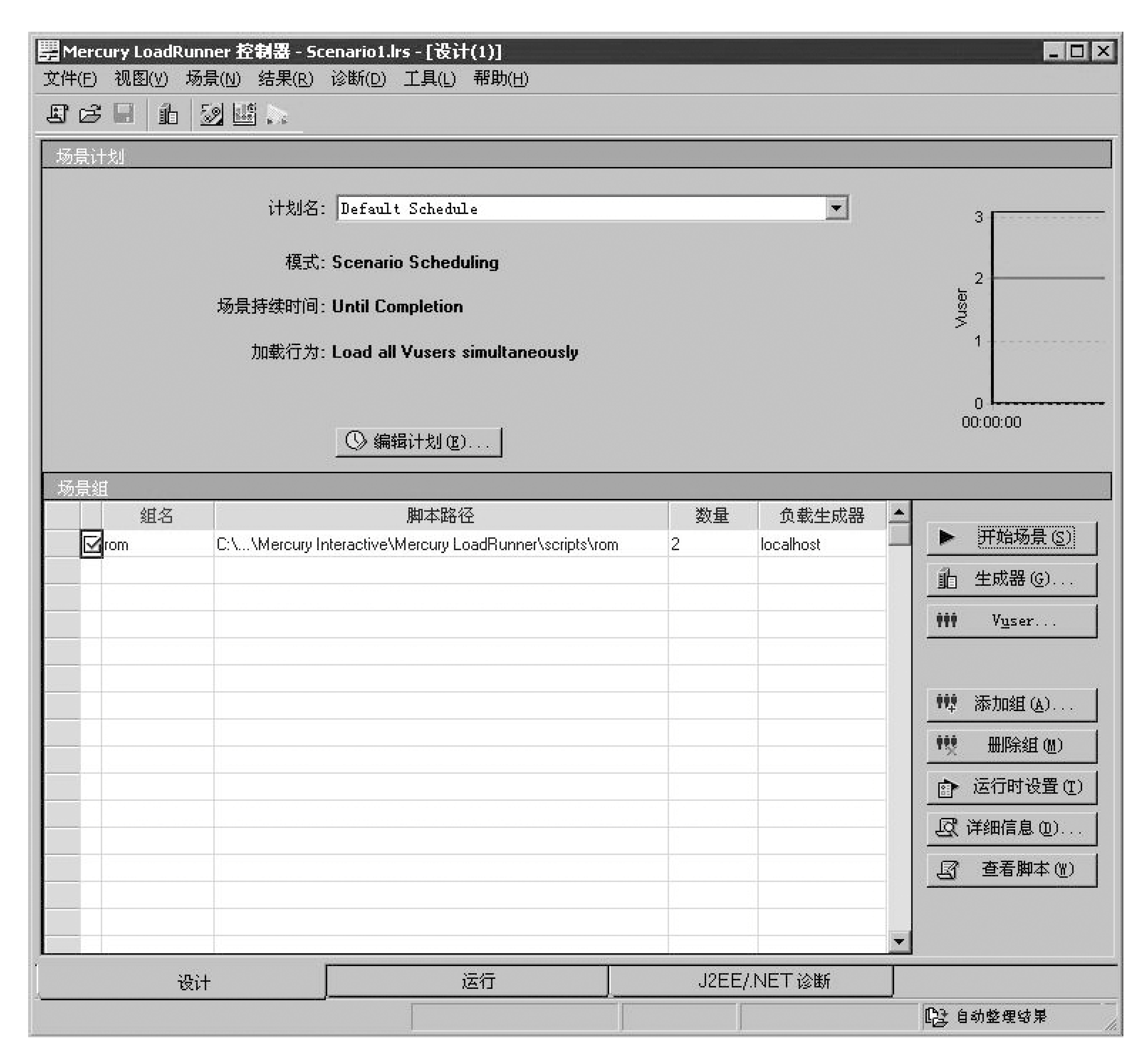


图 8-16　“Mercury LoadRunner 控制器”窗口

(1) 场景计划：可以设置负载行为以精确描绘用户行为。还可以确定将负载应用于应用程序的速率、负载测试持续时间以及如何停止负载。

(2) 场景组：配置 Vuser 组。可以创建代表系统典型用户的各种组。可以定义这些典型用户运行的操作、运行的 Vuser 数以及 Vuser 运行时所用的计算机。

8.6.2 负载测试

1. 创建负载测试

Controller 是用来创建、管理和监控测试的中央控制台。使用 Controller 可以模拟实际用户执行操作的示例脚本，并通过让多个虚拟用户同时执行这些操作以便系统中创建负载，执行过程如下[33,35]：

1) 打开"Mercury LoadRunner"窗口

选择"开始"→"程序"→ Mercury LoadRunner → LoadRunner，打开 Mercury LoadRunner Launcher 启动窗口。

2) 打开 Controller

在"负载测试"选项卡中，单击"运行负载测试"。默认情况下，LoadRunner Controller 打开时将显示"新建场景"对话框。

3) 打开示例测试

从 Controller 菜单中选择"文件"→"打开"，打开＜LoadRunner 安装＞\Tutorial 目录中的 demo_scenario. lrs。打开 LoadRunner Controller 的"设计"选项卡，demo_script 测试将出现在"场景组"窗格中。可以看到已分配两个 Vuser 运行测试(见图 8-16)。

2. 运行负载测试

单击"启动场景"按钮，显示 Controller 运行视图(见图 8-17)，Controller 将开始运行场景，在"场景组"窗格中，可以看到 Vuser 逐渐开始运行并在系统中生成负载。

3. 监控负载测试

使用 LoadRunner 的集成监控器套件可以度量负载测试期间每个单一层、服务器和系统组件的性能。LoadRunner 包括用于各种后端系统组件(包括 Web、应用程序、网络、数据库和 ERP/CRM 服务器)的监控器，执行过程如下[33]。

1) 查看默认图

默认情况下，Controller 显示正在运行的 Vuser 图、事务响应时间图、每秒点击次数图和 Windows 资源图。通过正在运行的 Vuser——整个场景图，可以监控指定时间正在运行的 Vuser 数。由图 8-18 可以看到 Vuser 以每分钟 2 个 Vuser 的速率逐渐开始运行。

通过事务响应时间——整个场景图，可以监控完成每个事务所花费的时间，可以看到客户登录、搜索航班、购买机票、查看线路和从系统注销所花费的时间，如图 8-19 所示。

可以看到随着越来越多的 Vuser 运行接受测试的应用程序，事务响应时间将增加，并且提供给客户的服务水平将降低。

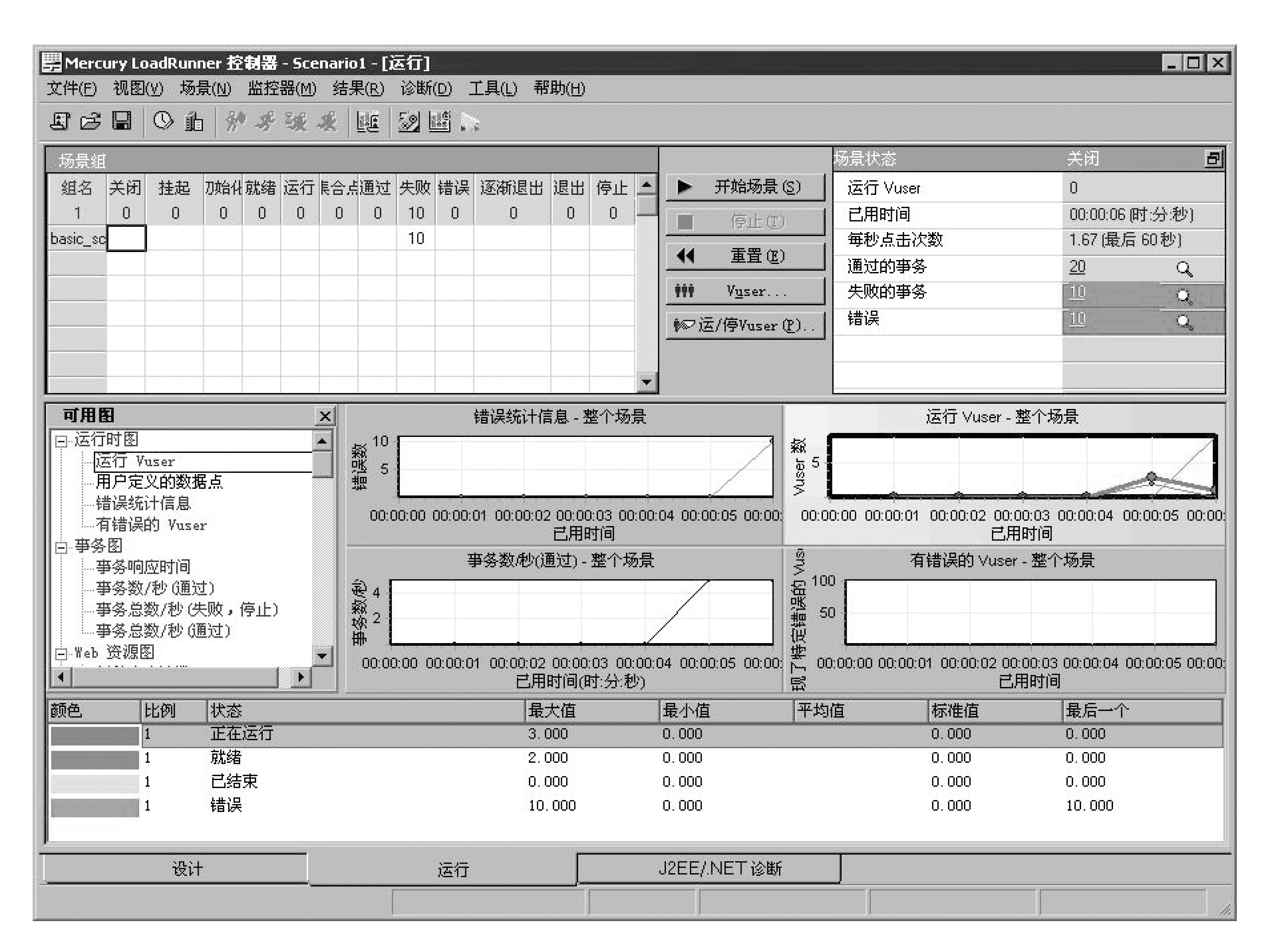


图 8-17 Controller 运行视图

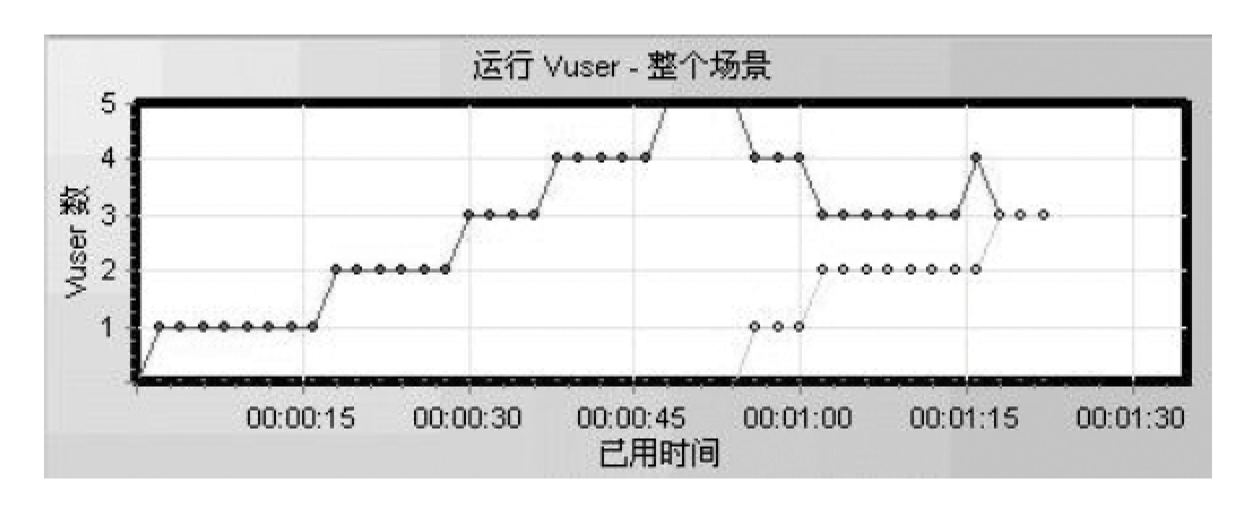


图 8-18 Vuser 图

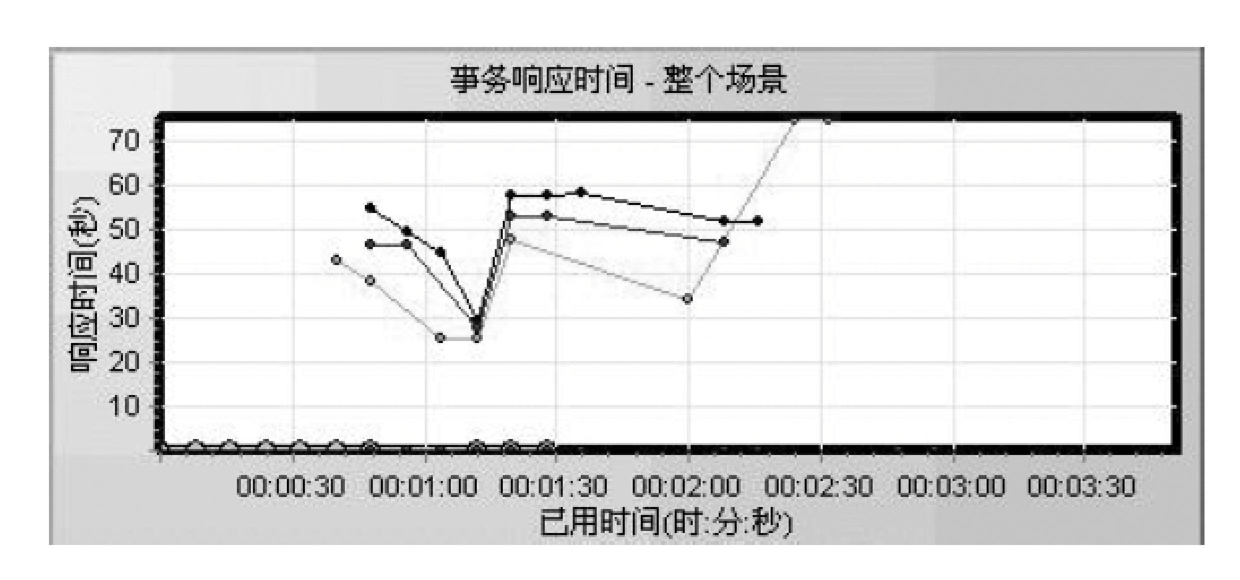


图 8-19 整个场景图

2）查看错误信息

如果计算机处理的负载很重，则可能遇到错误。在可用图树中选择错误统计信息图并将其拖入 Windows 资源窗格中。错误统计信息图提供了有关场景执行期间发生错误时间及错误数的详细信息。

在图 8-20 中，可以看到 5 分钟后系统开始遇到错误数不断增加。这些错误是由响应

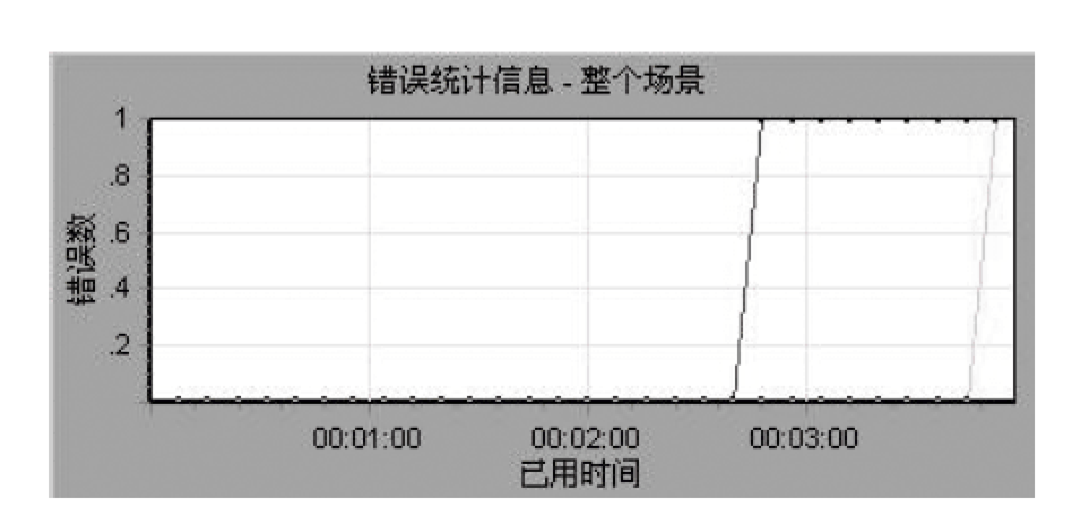


图 8-20　查看错误信息

时间降低引起的超时所导致的。

8.6.3　分析结果

测试运行结束时，LoadRunner 提供一个详细的图和报告。通过选择“结果”→“结果设置”或单击“分析结果”按钮，可以打开带有场景结果的分析。结果保存在＜LoadRunner 安装＞\Results\tutorial_demo_res 目录下。

分析(Analysis)会话过程中生成的图和报告提供了有关系统性能的重要信息。使用这些图和报告，可以轻松地标识和确定应用程序中的瓶颈以及提高系统性能所需的改进。通过运行 LoadRunner Analysis 以及生成和查看图及报告，有助于找出性能问题并确定问题的根源。

启动 Analysis 会话执行如下操作[33]：

1. 打开 Mercury LoadRunner

选择“开始”→“程序”→ Mercury LoadRunner → LoadRunner。打开 Mercury LoadRunner Launcher 窗口。

2. 打开 LoadRunner Analysis

在“负载测试”选项卡中，单击“分析负载测试”，打开 LoadRunner Analysis。

3. 打开 Analysis 会话文件

运行一个测试场景，在 Analysis 窗口中，选择“文件”→“打开”，打开“打开现有分析会话文件”对话框(见图 8-21)，或者在＜LoadRunner＞安装目录 Tutorial 文件夹中，选择 analysis_session 并单击“打开”，也将打开 Analysis 会话文件。

Analysis 窗口包括图树、图查看区域和图例三部分，如图 8-22：

(1) 图树：左窗格中分析显示可以打开查看的图，可以在此处显示打开分析时未显示的新图，或删除不再想查看的图。

(2) 图查看区域：在右窗格中显示分析情况。默认情况下，当打开一个会话时，分析概要报告将显示在此区域。

(3) 图例：位于底部窗格中可以查看选定图中的数据。

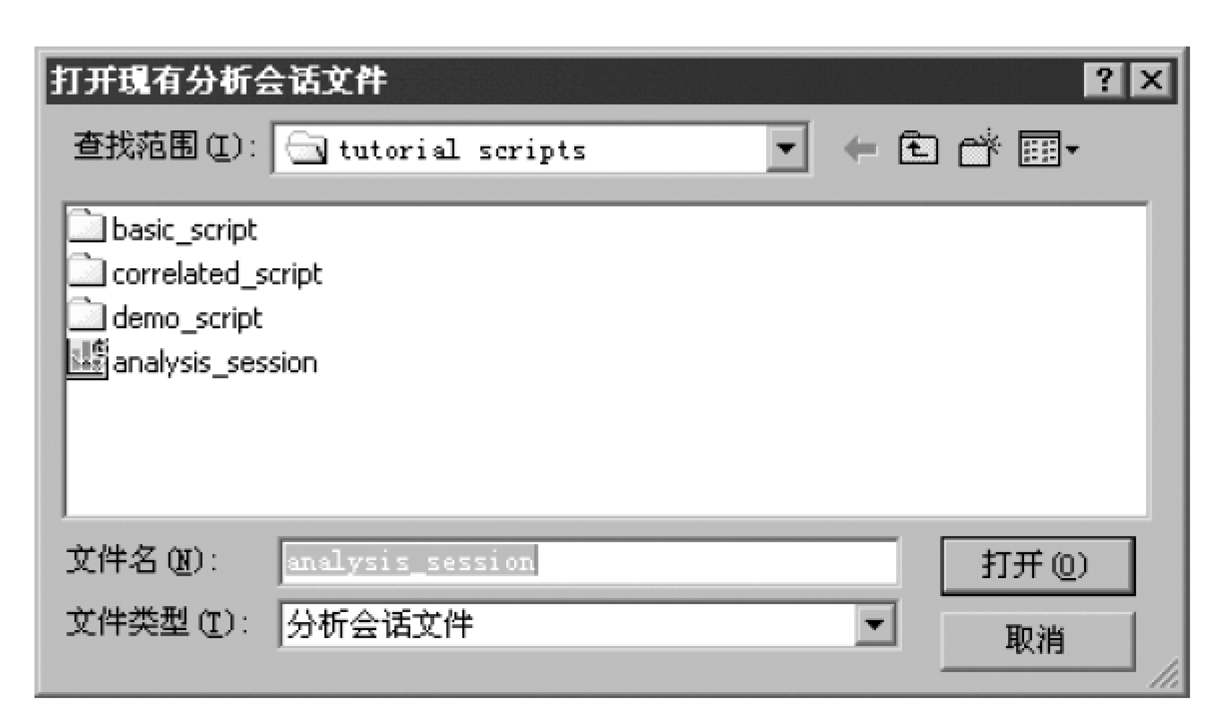


图 8-21 “打开现有分析会话文件”对话框

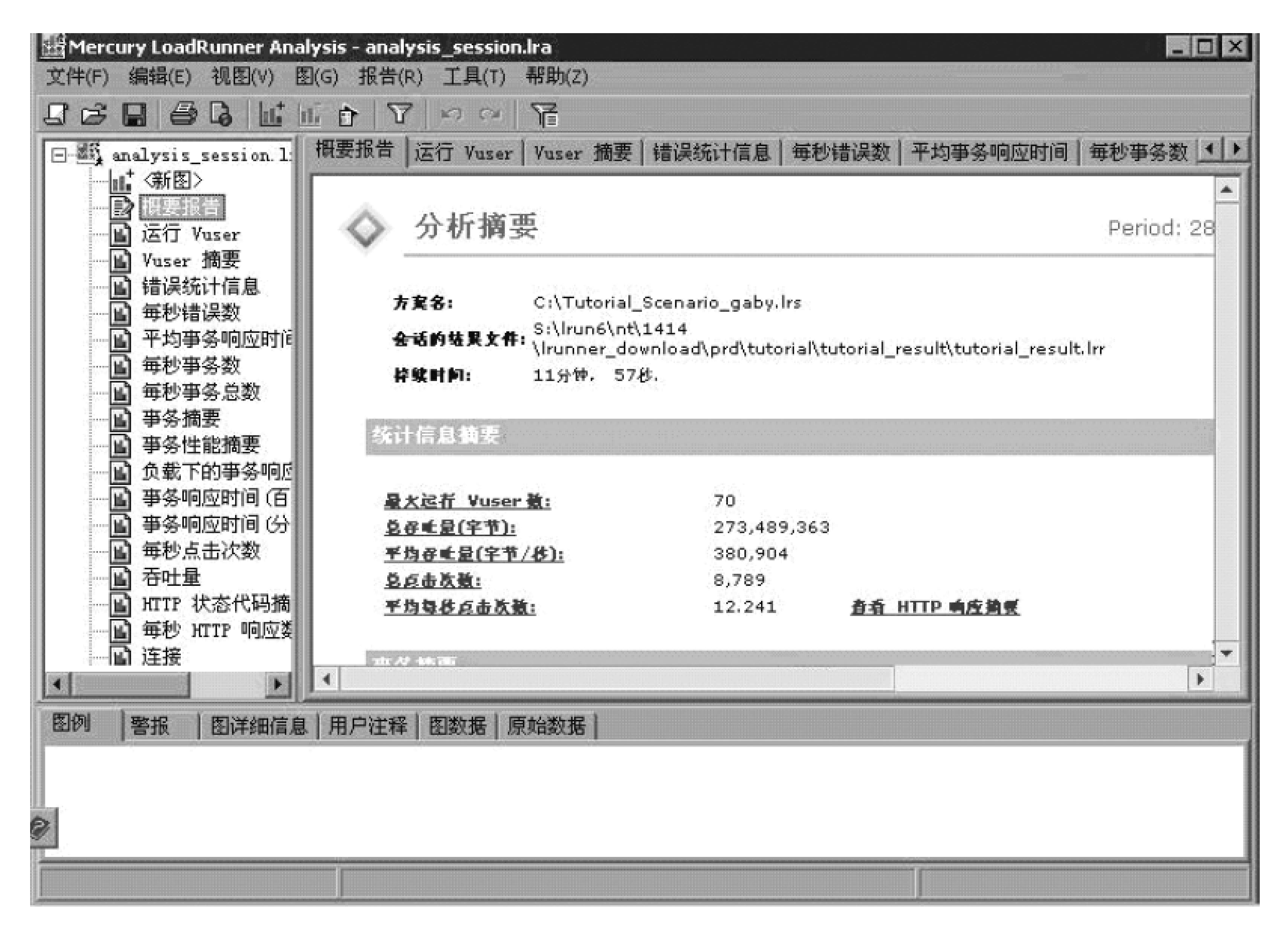


图 8-22 Analysis 窗口

8.7 系统性能测试

LoadRunner 内含集成的实时监测器，能够实时显示交易性能数据，评估系统组件性能，以便及时发现问题。

8.7.1 Run-Time Setting 配置

完善测试脚本后，对 VuGen 的 Run-Time Setting 进行配置。下面对经常需要设置的标签页进行说明。首先打开 Run-Time Setting 窗口，可以通过菜单或者工具栏进行。操作过程如下：

(1) 选择 Vuser→run-time settings，出现 Run-time Setting 对话框，如图 8-23 所示。测试过程中 Run-time Setting 对话框各选项根据具体要求做相应设置。



图 8-23 “Run-Time Setting”对话框

（2）切换到 Network 选项卡，如图 8-24 所示。带宽越大，Web 服务器压力越大。

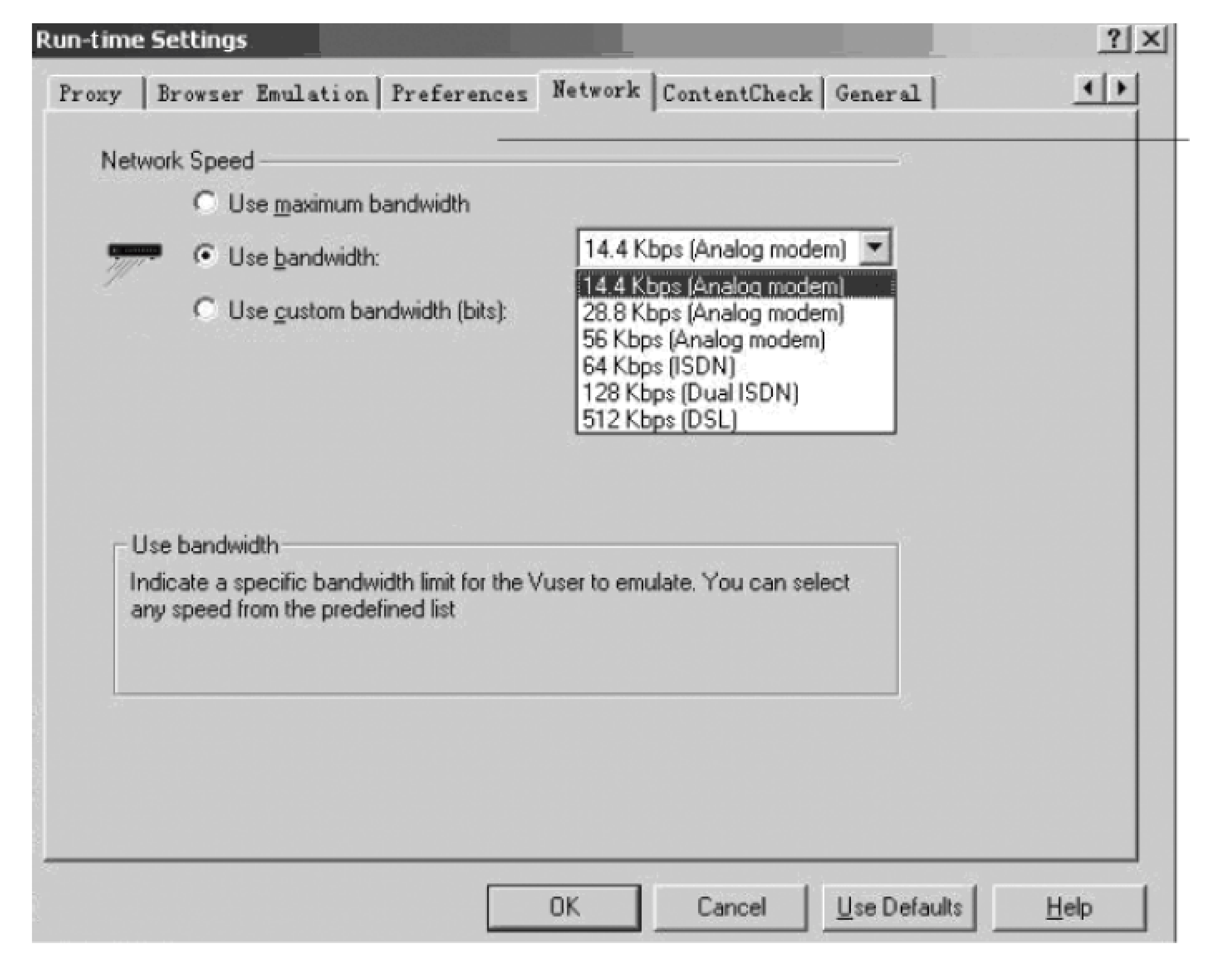


图 8-24 Network 选项卡

（3）切换到 Preferences 选项卡，如图 8-25 所示。

如切换到 ContentCheck 选项卡，能够检测何种页面为错误界面。如果服务器返回的界面包含预定义的字符串，则检测到错误页面 VuGen 停止运行，指示失败。

8.7.2 监控负载下的应用程序

使用 Controller 联机图查看监控器收集的性能数据，通过该信息隔离系统环境中可

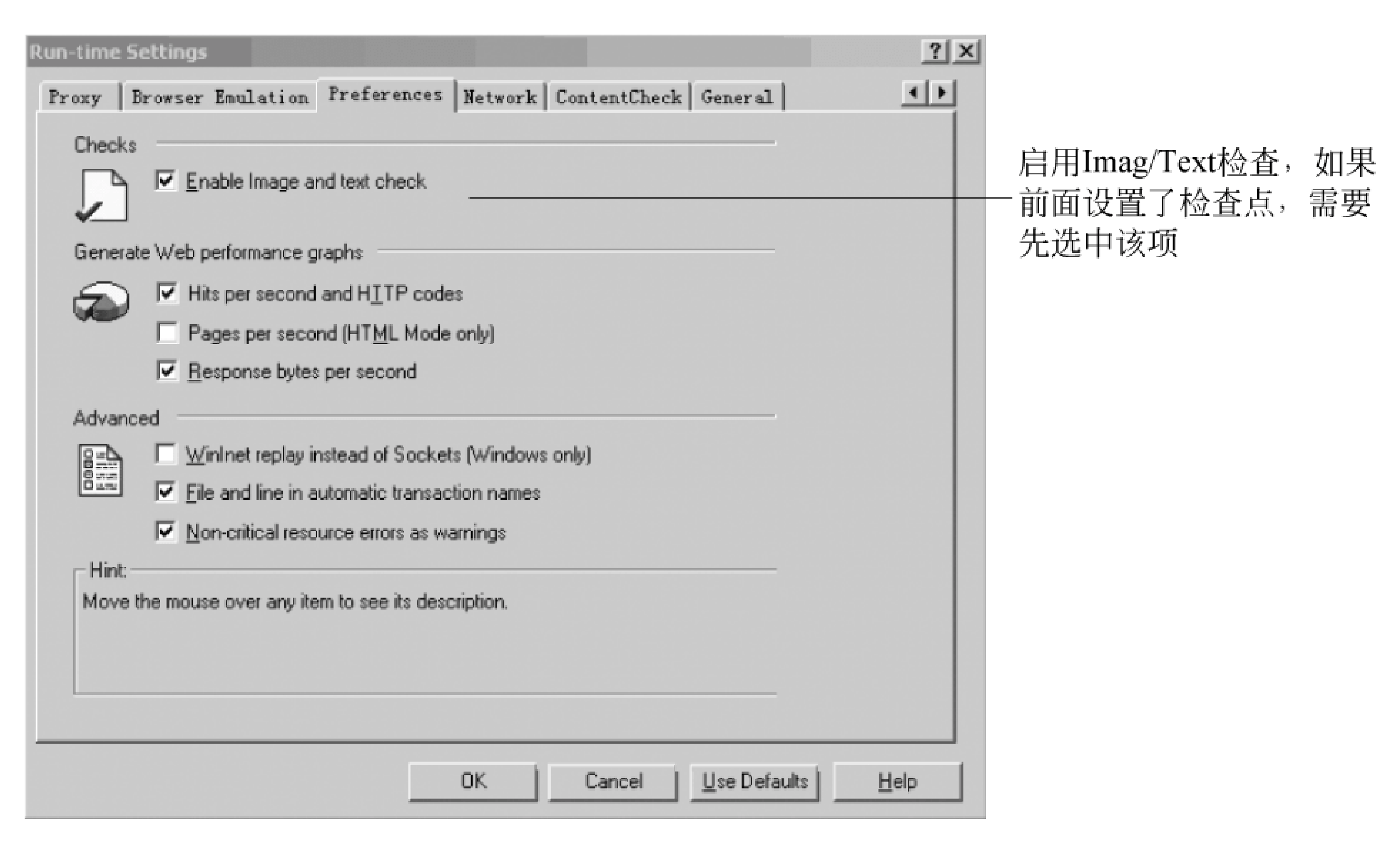


图 8-25　Preferences 选项卡

能存在问题的区域，执行操作如下[33]。

1. 检查性能图

单击 Controller 的"运行"选项卡，将显示整个场景对应不同性能条件的监控曲线（见图 8-26）。

- 正在运行的 Vuser——整个场景图，显示指定时间正在运行的 Vuser 数。
- 事务响应时间——整个场景图，显示完成每个事务所需的时间。
- 每秒点击次数——整个场景图，显示场景运行每一秒内 Vuser 在 Web 服务器上的点击次数。
- Windows 资源图，显示场景运行期间度量的 Windows 资源。

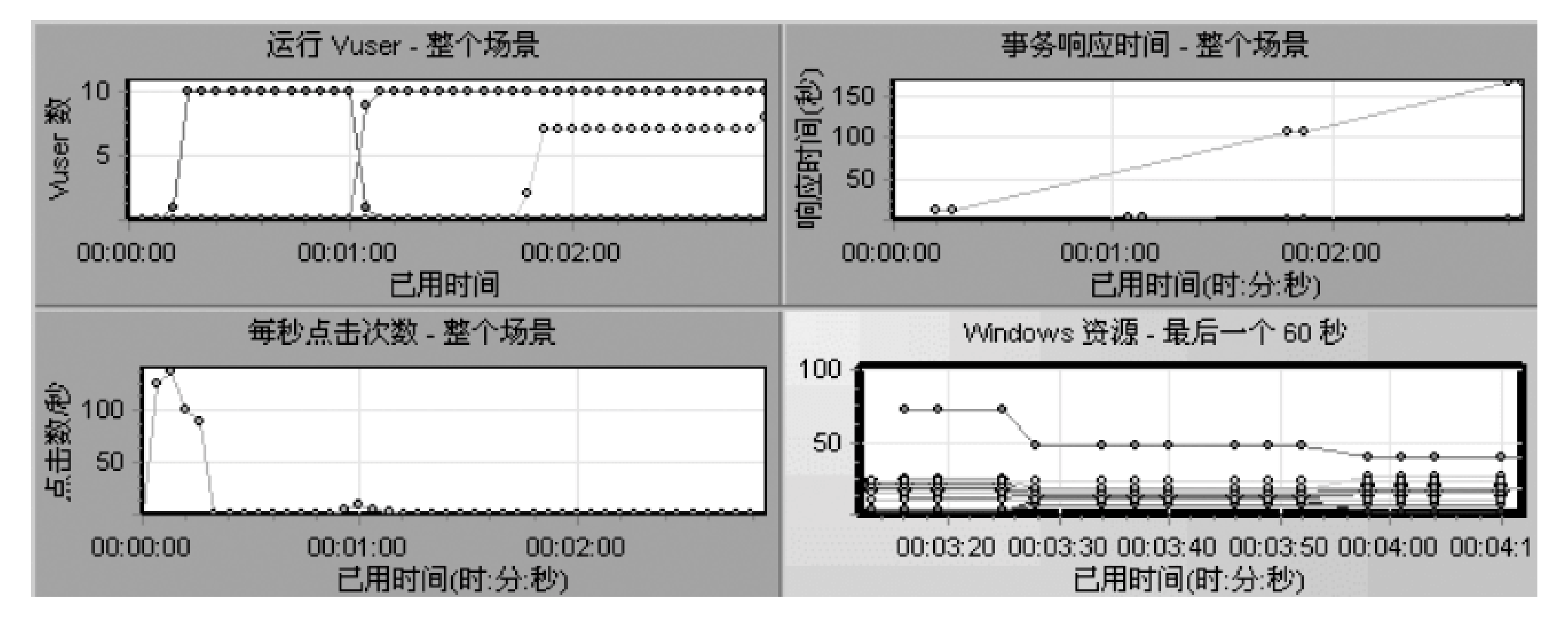
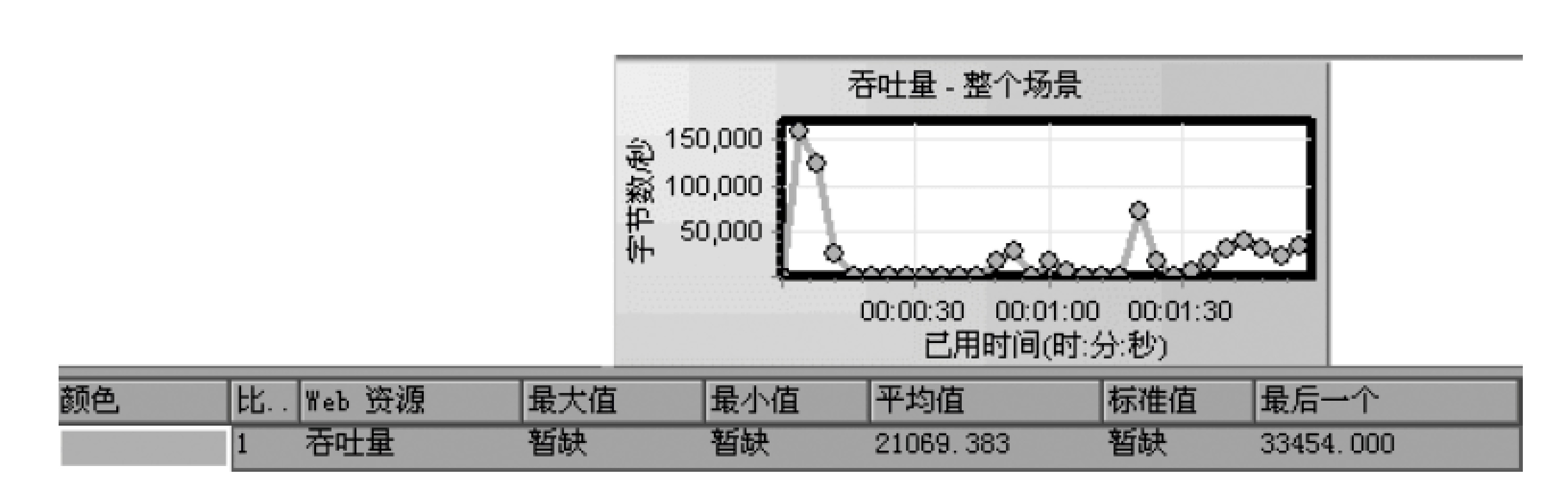


图 8-26　各种运行情况——场景图

2. 查看吞吐量信息

在可用图树中选择吞吐量图并将其拖入图查看区域。吞吐量（见图 8-27）显示 Vuser 在任何给定的某一秒上从服务器接收到的数据量（度量单位是字节）。将图 8-27 与图 8-26

中的事务响应时间图进行比较，查看吞吐量对事务性能产生的影响。如果吞吐量随着时间推移和 Vuser 数量增加而上升，这表明带宽足够宽。如果随着 Vuser 数量的增加图 8-27 曲线保持相对平滑，说明带宽制约了传送的数据量。



颜色	比..	Web 资源	最大值	最小值	平均值	标准值	最后一个
	1	吞吐量	暂缺	暂缺	21069.383	暂缺	33454.000

图 8-27　吞吐量——场景图

8.7.3　测试期间增加负载

可以通过手动添加更多 Vuser，要在负载测试期间增加负载，执行下列操作：

(1) 单击“运行”/“停止”按钮，打开“运行/停止 Vuser”对话框(见图 8-28)，其中显示当前分配的在场景中运行的 Vuser 数。

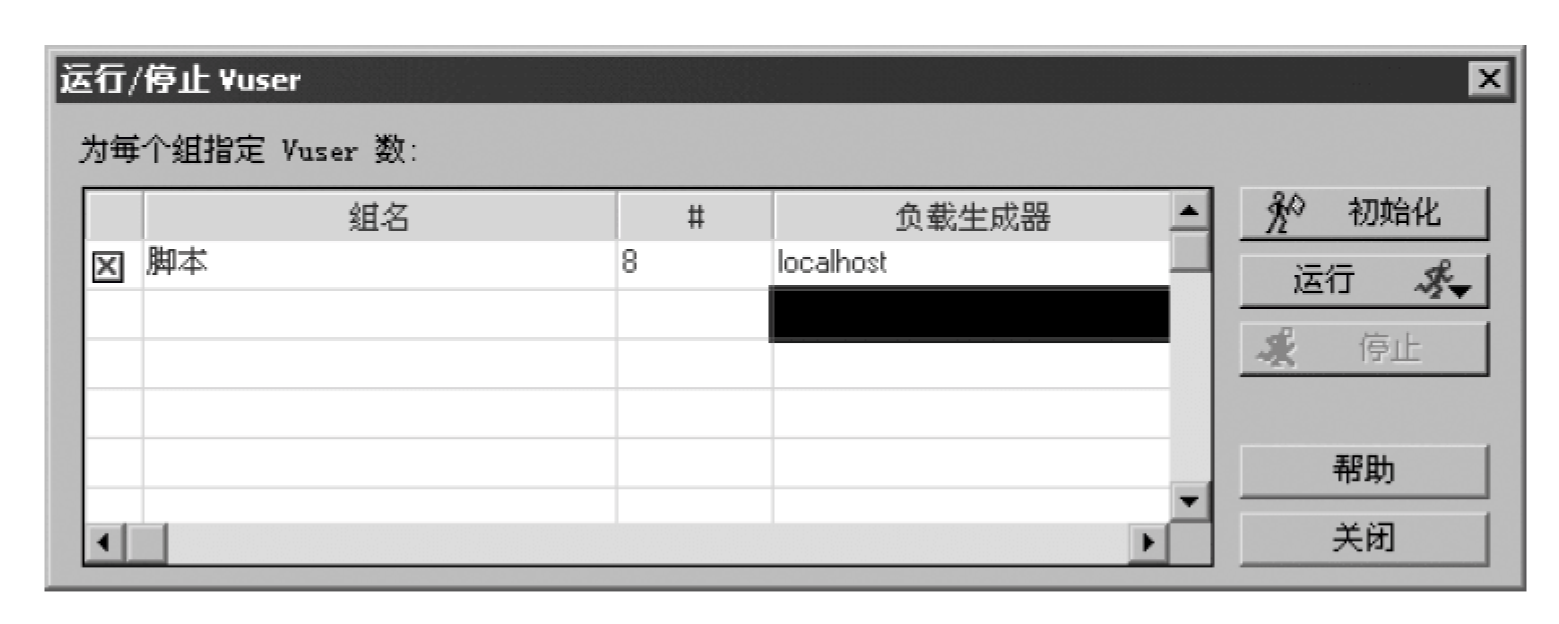


图 8-28　“运行/停止 Vuser”对话框

(2) 单击“运行”添加 Vuser，如果某些 Vuser 尚未初始化，单击“运行选择”打开“运行新 Vuser”选项，添加 Vuser。

测试运行结束时，“场景状态”窗口将显示“关闭”状态。这表示 Vuser 已停止运行。保存场景可以在相同的设置下再次运行它，选择“文件”→“保存”或单击“保存”按钮，然后在“文件名”中输入场景的名称即可。如果查看应用程序在负载下的执行情况，需要查看事务响应时间并确定事务是否位于客户可接受的范围内。如果事务响应时间延长，则需要查找瓶颈。完成调整后，重复运行负载测试以确认这些调整是否达到预期的效果，重复此循环优化系统性能。

第 9 章



测试流程自动化

9.1 JIRA 介绍

JIRA 是目前比较流行的基于 Java 架构的问题跟踪系统，可以帮助技术人员进行缺陷的管理。同时，充分利用 JIRA 的灵活配置和扩展性，可以将 JIRA 作为一个项目管理系统或者 IT 支持系统。由于 Atlassian 公司对很多开源项目实行免费提供缺陷跟踪服务，所以在开源领域，其认知度比其他的产品要高得多，而且易用性也好一些。同时，开源则是其另一特色。在用户购买其软件的同时，也就将源代码也购置进来，方便做二次开发。JIRA 功能全面，界面友好，安装简单，配置灵活，权限管理以及可扩展性较好。

9.1.1 JIRA的主要功能

JIRA 不仅是一个缺陷跟踪系统，拥有较人性化的用户界面，而且可以整合客户、开发人员、测试人员。其中，Eclipse 和 IDEA 下的 JIRA 插件，主要为开发人员服务，实时将信息反馈给开发人员，具备非常灵活的邮件通知配置。开发人员同时迅速地将修复的结果信息反馈到跟踪系统中，最后通过持续集成，软件迅速地完成更新。这些便捷的操作会极大地鼓舞软件开发中的各方人员，甚至包括客户。跟同类软件产品 TestTracker、ClearQuest、TestDirector 相比，JIRA 的性价比最好。其主要功能包括[36]：

(1) 问题追踪和管理。用它管理项目，跟踪任务、bug、需求，通过 JIRA 的邮件通知功能进行协作通知，在实际工作中使工作效率提高很多。

(2) 问题跟进情况的分析报告。可以随时了解问题和项目的进展情况。

(3) 项目类别管理功能。可以将相关的项目分组管理。

(4) 组件/模块负责人功能。可以将项目的不同组件/模块指派相应的负责人，来处理所负责的组件的问题(Issues)。

(5) 项目 E-mail 地址功能。每个项目可以有不同的 E-mail(该项目的通知邮件从该地址发出)。

(6) 无限制的工作流。可以创建多个工作流为不同的项目使用。

然而，JIRA 对于测试需求、测试用例等都没有提供直接的方式进行管理。不过可以利用 JIRA 的 Issue Type 的可定制性，来进行需求和测试用例方面的管理，也可以与 Testlink 集成。

9.1.2 JIRA版本说明

JIRA 4.0 前的版本按照功能分标准版、专业版、企业版。JIRA 4.0 版本之后，按照用户数量来划分：25、50、100、无限制用户，所有的版本都具有之前企业版的功能。JIRA 不限制创建项目数和 Issue 的数量，购买之后可以永久使用，并且一年内免费更新。目前，JIRA 官方网站上发布的最新版本为 JIRA 4.3.3。

9.1.3 JIRA涉及的角色

JIRA 作为一个缺陷跟踪管理系统，可以被企业管理人员、项目管理人员、开发人员、分析人员、测试人员和其他人员所广泛使用。JIRA 中涉及的角色包括管理人员、项目管理者、开发人员和测试人员。每个角色的职责如下。

1. 管理人员

根据 JIRA 系统提供的数据，更加准确地了解项目的开发质量和状态，以及整个团队的工作效率。

2. 项目管理者

可以针对登记进 JIRA 系统中问题，进行评估，分配缺陷；还可以通过 JIRA 系统的统计报告了解项目进展情况以及团队的工作量、工作效率等信息。

3. 开发人员

在 JIRA 系统中查看分配给自己的任务，及时处理，填写处理情况并提交工作量记录。

4. 测试人员

根据测试情况，在 JIRA 系统中及时快速地记录问题并对开发人员处理后的问题进行验证和跟踪。

9.2 JIRA的概念

9.2.1 问题

JIRA 跟踪问题(issue)。这些问题可以是 bug，功能请求或者任何其他想要跟踪的任务。每一个问题有相关联的问题类型(issue type)、摘要(summary)、问题描述(description)、问题所属的项目、问题关联的项目组件(component)、问题影响的项目版本(affect version)、问题将被解决的项目版本(resolved version)、问题发生的环境、问题的优先级、问题的报告者、问题的指派处理人、问题的当前状态和问题相关的历史记录等信息。

1. 问题类型

JIRA 系统可以用于跟踪多种不同类型的问题。系统管理员可以根据需要添加。JIRA 系统默认提供的问题类型如下：

- Bug：测试过程、维护过程发现影响系统运行的缺陷。
- New FeaTrue：对系统提出的新功能。
- Task：需要完成的任务。
- Improvement：对现有系统功能的改进。

2. 优先级

在 JIRA 系统中用优先级(Priority Levels)来表示问题的严重级别。系统管理员可以在 JIRA 系统中添加优先级，JIRA 系统默认的优先级如表 9-1 所示。

表 9-1　JIRA 系统默认优先级

级别	参考描述
Blocker	阻塞开发或测试的工作进度，或影响系统无法运行的错误
Critical	系统崩溃，丢失数据或内存溢出等严重错误、或者必须完成的任务
Major	主要的功能无效、新增功能建议
Minor	功能部分无效或对现有系统的改进
Trivial	拼写错误，文本未对齐等

3. 状态

每个问题用状态(statas)来表明问题所处的阶段。问题依次经过“开始(Open)”、“处理(Progress)”、“解决(Resolved)”、“被关闭(Closed)”状态。可以根据项目来定制问题状态以及工作流。JIRA 系统提供的默认状态如表 9-2 所示。

表 9-2　JIRA 系统提供的默认状态

默认状态	参考描述
Open	表示问题被提交等待有人处理
In Progress	问题在处理当中，尚未完成
Resolved	问题曾解决，但解决结论未获认可，需要重新分派解决
Reopened	问题解决，等待结果确认，确认的结果是 Reopened 或者 Closed
Closed	问题处理结果确认后，置于关闭状态

4. 解决

一个问题可以用多种方式解决(Resolutions)。系统管理员可以在 JIRA 中定制解决方式。JIRA 默认的解决方式有：

- Fixed：已经解决的。
- Won't Fix：未解决，将不会解决的。

- Duplicate：重复的。
- Incomplete：描述得不够准确、完全的。
- Cannot Reproduce：重现失败，或者无足够信息重现的。

9.2.2 项目

一个 JIRA 通常包含许多项目，这些项目相当于产品或者开发项目。每一个问题属于一个项目，每一个项目有一个名字和一个关键字（如 Web），以后属于这个项目的问题的关键字就会包含 Web（如 Web-100、Web-101 等）。值得注意的是，在 JIRA 中有一个权限 Administer Projects。通常将这个权限赋给项目负责人，拥有这个权限的 JIRA 用户就可以管理项目的"版本"和"组件"。

一个项目一般会有多个版本，如：1.0alpha、1.0beta、1.0、1.2、2.0。JIRA 中的问题涉及到两个版本字段：影响版本和修复版本。一个 bug 可能会有影响版本 1.1 和 1.2，修复版本 1.1 和 1.2，可能在 2.0 版本上被解决。版本通常有"发布（Released）"、"未发布（Unreleased）"和"归档（Archived）"三种状态。版本还有发布日期，在特定的报告中会显示。在 JIRA 中可以为项目创建版本。

每一个项目通常会包含多个组件/模块，如后台、GUI、邮件子系统等。在 JIRA 系统中可以为项目添加组件。

一般情况下，会把某个问题分配给某个团队成员去解决。对问题的说明可以通过上传附加文件或屏幕截图。

9.3 JIRA 的应用

9.3.1 安装与配置

JIRA 支持各种主流的 Web 浏览器，如 IE、Mozilla、Firefox、Opera、Safari 等。JIRA 的运行需要 Java 环境。Java 是一个跨平台的编程语言，因此所有支持 Java 的操作系统都可以运行 JIRA。当然也需要考虑系统上是否支持您选择的应用服务软件。

JIRA 的安装需要下载 JDK、JIRA 相关的版本。JDK 的安装本章节不再详述，JIRA 的安装较容易，只要按照安装向导进行安装即可。另外，还可以为 JIRA 配置单独的数据库，如 MYSQL、MS SQL、ORACLE 等。

配置 JIRA，需要在浏览器中输入：http://localhost:8080，按 Enter 键进入配置界面，然后根据浏览器界面上的 JIRA 配置向导，经过三个步骤完成配置。

（1）配置 JIRA 系统的属性，如图 9-1 所示。

图中，在"＊程序标题"文本框中输入安装 JIRA 系统的标题信息；在"＊模式"的下拉列表框中选择 Public 或 Private 模式。其中，Public 模式下用户可以自己注册到 JIRA 系统中，并可以创建问题。Private 模式下不允许用户随意的注册，只能由 JIRA 系统管理员来创建用户、分配权限；在"＊根地址"中输入 JIRA 的访问地址，允许用户在浏览器中



图 9-1　JIRA 系统配置步骤(1)

通过该 URL 来访问 JIRA;在“* 索引路径”中输入保存 JIRA 系统索引数据的目录,如 C:\jira\index;在 Enable Backups 中可选择是否启用 JIRA 系统自动备份功能;“附件路径”中输入保存与问题关联的附件文件的目录,如 D:\jira\attachments;“备份路径”中保存 JIRA 备份文件的目录,如 D:\jira\backups,用于自动存储 JIRA 定期自动备份的文件;“授权码”中输入评估 license,如果没有的话,可以单击“在线”连接生成一个评估 license;单击“下一步”按钮进入下一步骤。

(2) 配置 JIRA 系统管理员的信息,如图 9-2 所示。

输入相应的用户名、密码及邮件等信息,单击“下一步”按钮进入下面的步骤。

(3) 配置 JIRA 系统的邮件通知参数,如图 9-3 所示。

输入相应的参数完成邮件通知配置,也可以选择禁止邮件通知。经过上述三大步骤,JIRA 系统配置完成。

9.3.2　登录和注册

在成功安装配置完成后,输入用户名和密码,就可以登录 JIRA 系统。也可单击“注册”按钮注册另外的用户账号。JIRA 系统的登录界面和注册账号界面分别如图 9-4 和图 9-5 所示。

JIRA 安装

步骤2(共3): 管理员帐号

设置初始管理员的帐号. 其他的管理员可以以后添加.

*用户名: violet wang

*密码: ••••••

*确认: ••••••

*全称: violet wang

帐号的全称(例如 Joe Citizen).

*邮件:

管理员的邮件地址

下一步>>

图 9-2　JIRA 系统配置步骤(2)

步骤3(共3): 邮件通知

设置邮件服务器用来发送邮件通知,或者关闭邮件通知.

禁止邮件通知

名称: Default SMTP Server

服务器的名称.

来自的邮件地址:

用来发送邮件的缺省邮件地址 (确认这个邮件可以在邮件服务器上发送邮件).

邮件前缀: [JIRA]

前缀添加在每个邮件主题的前面 (帮助用户过滤邮件).

服务器详情

输入SMTP服务器的预定或者输入javax.mail.Session对象的JNDI 地址.

SMTP Host

网址:

如果想通过SMTP服务器发送邮件通知, 输入服务器的域名

Protocol: SMTP

SMTP 端口:

可选 - SMTP 端口. 不填就使用缺省的端口(缺省: 25).

Timeout: 10000

Timeout in milliseconds - 0 or negative values indicate infinite timeout. Leave blank for default (10000 mSecs).

TLS:

Optional - the mail server requires the use of TLS security.

用户名:

可选 - 如果你的服务器发送邮件需要认证,请输入用户名.

密码:

可选 - 同上, 输入对应的密码.

或

JNDI 地址

JNDI 地址:

如果通过存在的mail session发送邮件通知, 输入JNDI 地址(javax.mail.Session).

Test Connection　完成

图 9-3　JIRA 系统配置步骤(3)

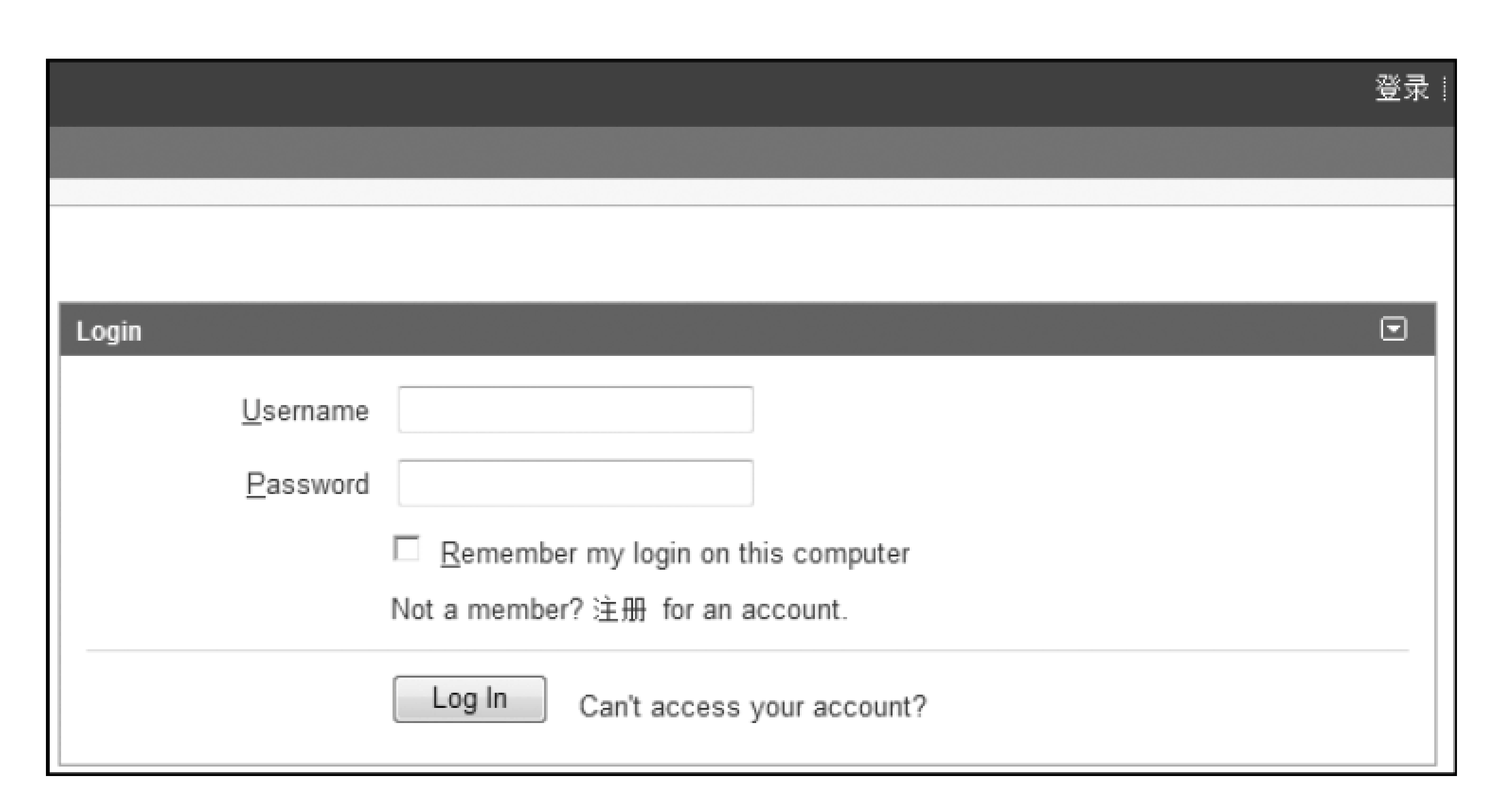


图 9-4　JIRA 系统登录界面

注册

请输入你的详细信息.

用户名*

密码*

确认密码*

全名*

邮件*

注册 Cancel

图 9-5 JIRA 系统注册账号界面

9.3.3 创建新项目

通常，开始使用 JIRA 首先应创建项目。单击“现在创建一个项目”，弹出添加新项目界面，如图 9-6 所示。输入项目的名称、关键字/Key、项目负责人、描述信息、通知模型(Notification Scheme)、权限模型(Permission Scheme)。Notification Scheme 和 Permission Scheme，可以分别选择 Default Notification Scheme 和 Default Permission Scheme(默认项)。最初使用时可以暂时不选择 Issue Security(网址和问题安全等级)选项。

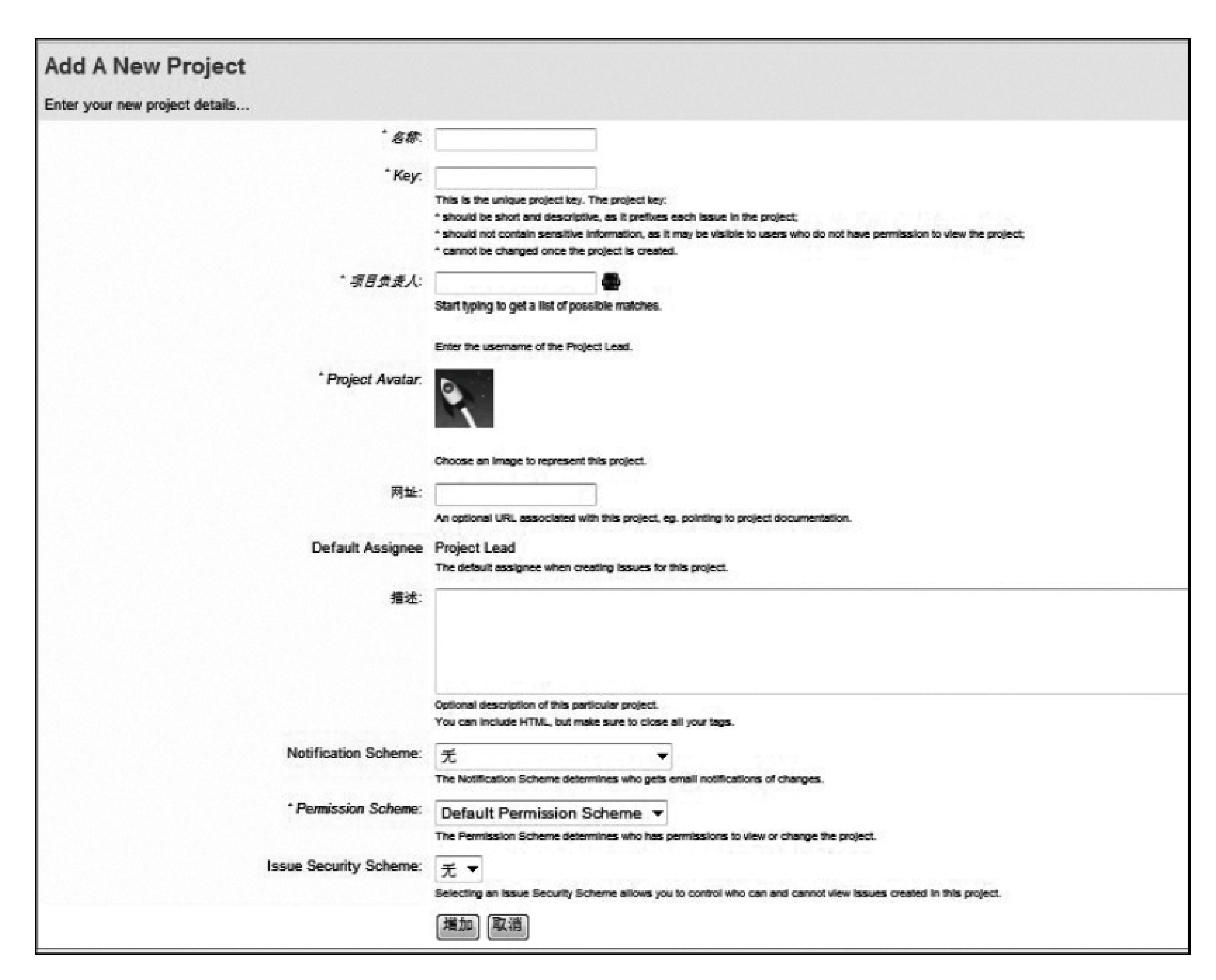


图 9-6 添加新项目界面

项目的信息填写完成后，单击“增加”按钮就能看到在 JIRA 系统中创建的新项目的详细信息界面，如图 9-7 所示。

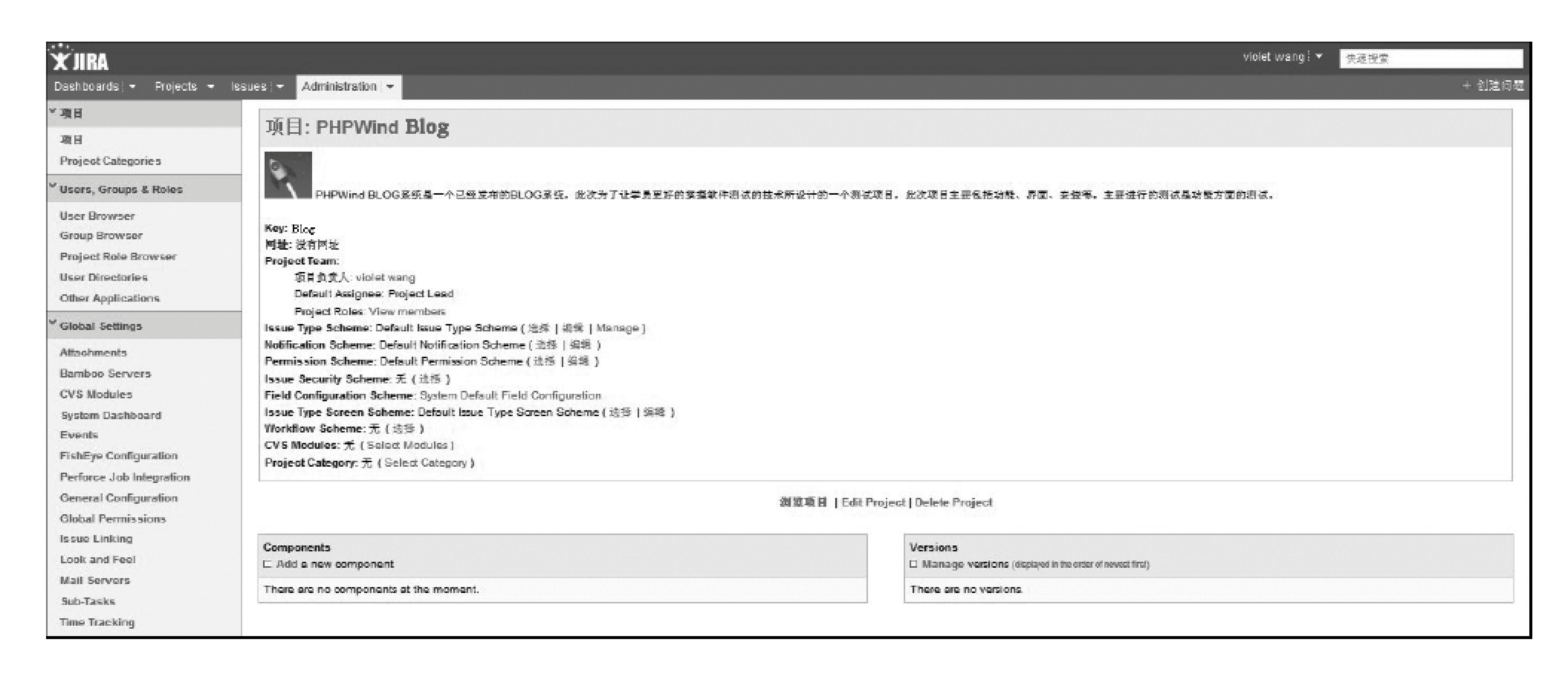


图 9-7　添加好的项目基本信息

在项目的详细信息界面上，可选择 Components 下方 Add a new Component 的 Add 链接，创建项目的组件或模块，可选择 Versions 下面 Manage Versions(displayed in the order of newest first)的 Manage 链接管理项目的版本，如图 9-8 所示。

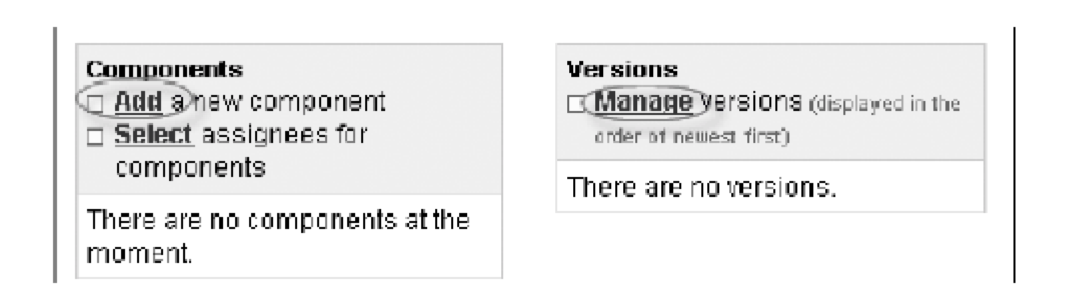


图 9-8　创建组件界面图

注意：新项目创建完成后，项目的关键字(key)是不能修改的。如果读者想要继续创建新项目，则可以单击左侧的“项目”链接，然后再单击 Add Project 链接。

9.3.4　创建项目类别

在使用 JIRA 过程中，会有越来越多的项目被添加到 JIRA 系统中。这时可以利用 JIRA 系统的项目类别(Project Categories)功能，定义一些项目类别名称，然后将同类的项目归到一个类别里面，方便管理。项目类别添加界面如图 9-9 所示。

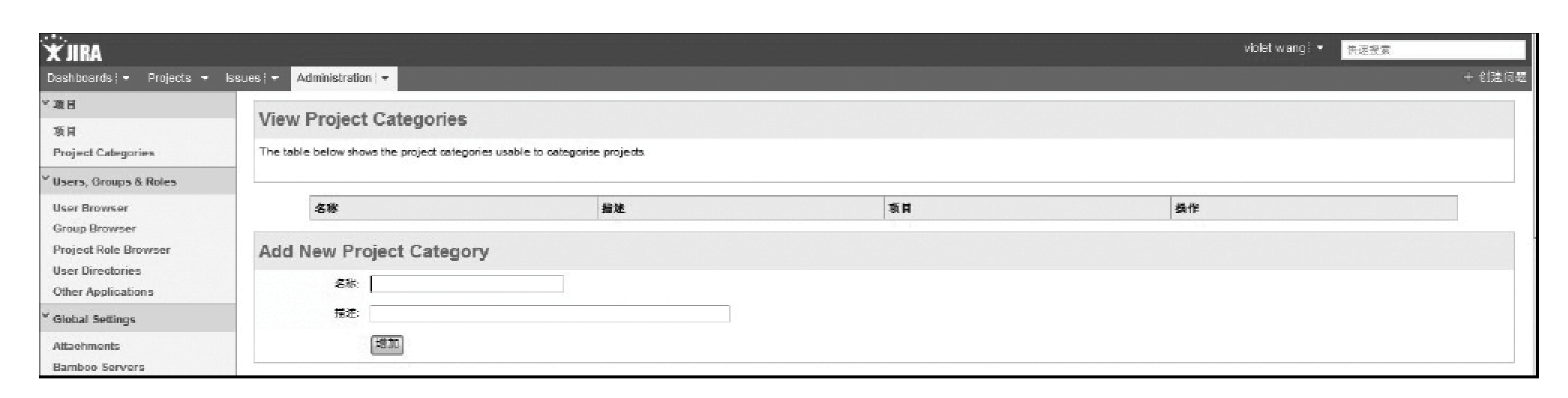


图 9-9　项目类别添加界面

在图 9-7 的项目详细信息界面上找到“Project Category：无(Select Category)”，单击 Select Category 链接，选择一个项目类别，这样就把该项目添加到所选择的项目类别里了。

9.3.5 添加用户和组

1. 添加用户

在安装完 JIRA 后，系统中只有一个系统管理员账号。在创建完项目后，通常需要在 JIRA 中添加其他的用户账号。在管理界面上单击左侧的 Users，Groups&Roles→User Browser 链接，再单击 Add User 链接添加新用户，如图 9-10 所示。

注意：不要使用汉字作为用户名。

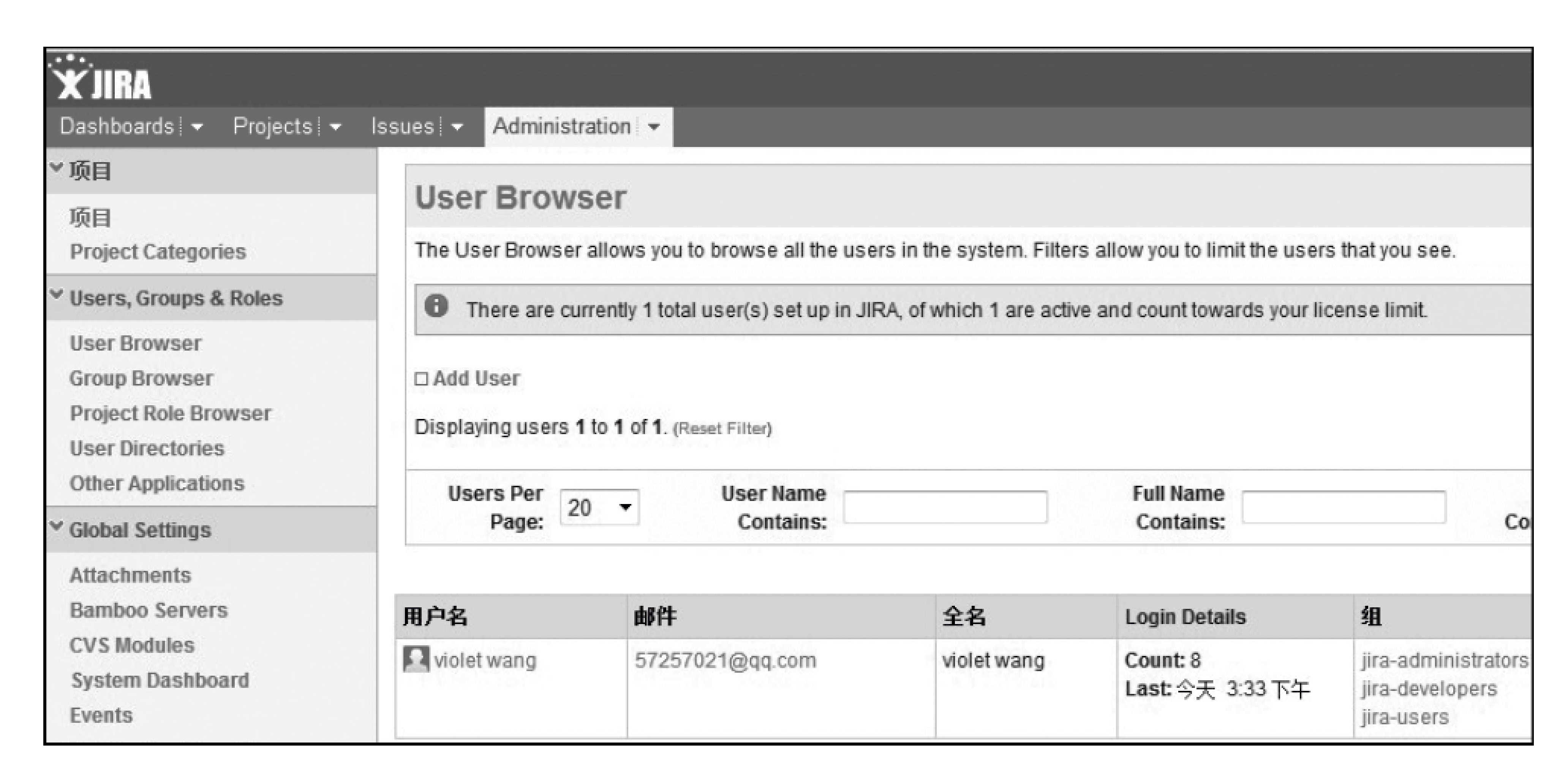


图 9-10 添加用户界面

JIRA 的用户是通过 Group(组)来管理的，通过给组授权，达到管理用户的目的。

2. 添加组

在管理界面上单击左侧的 Users&Groups→Group Browser 链接，在 Group Browser 界面右侧“名称”中输入组名，然后单击 Add Group 按钮完成组的添加，如图 9-11 所示。

注意：不要使用汉字作为组名。

9.3.6 创建问题

创建问题需要经过以下几个步骤。

① 选择项目和问题类型。单击图 9-11 右上角的“创建问题”链接。

② 填写或选择创建的问题详细信息。主要包括：问题的概要描述、优先级、逾期日期、所属模块、问题影响的版本、问题修复的版本、分配给哪个人员、问题出现的环境描述、问题详细信息描述。以 PHPWind BLOG 项目为例，填写或选择创建问题的详细信息，如图 9-12 所示。

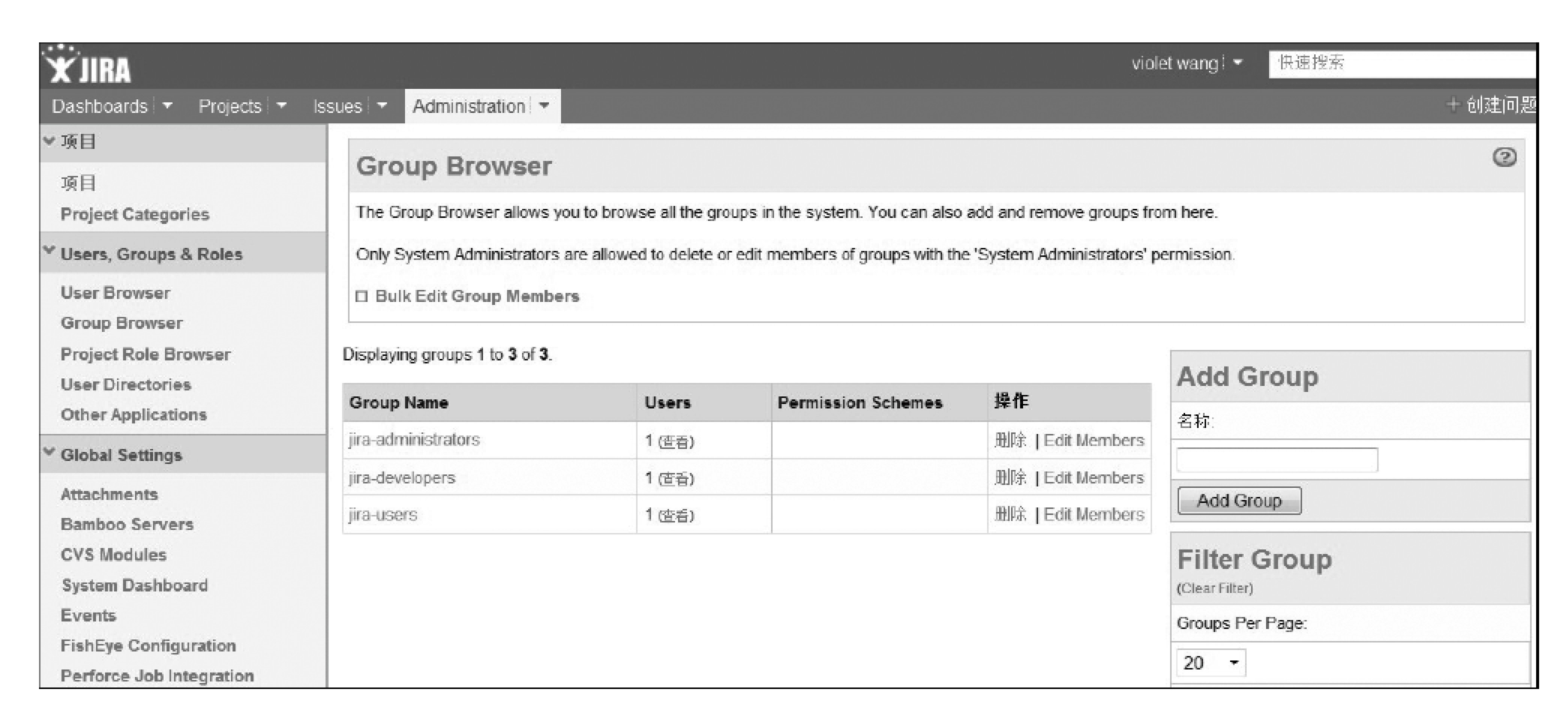


图 9-11　添加组界面

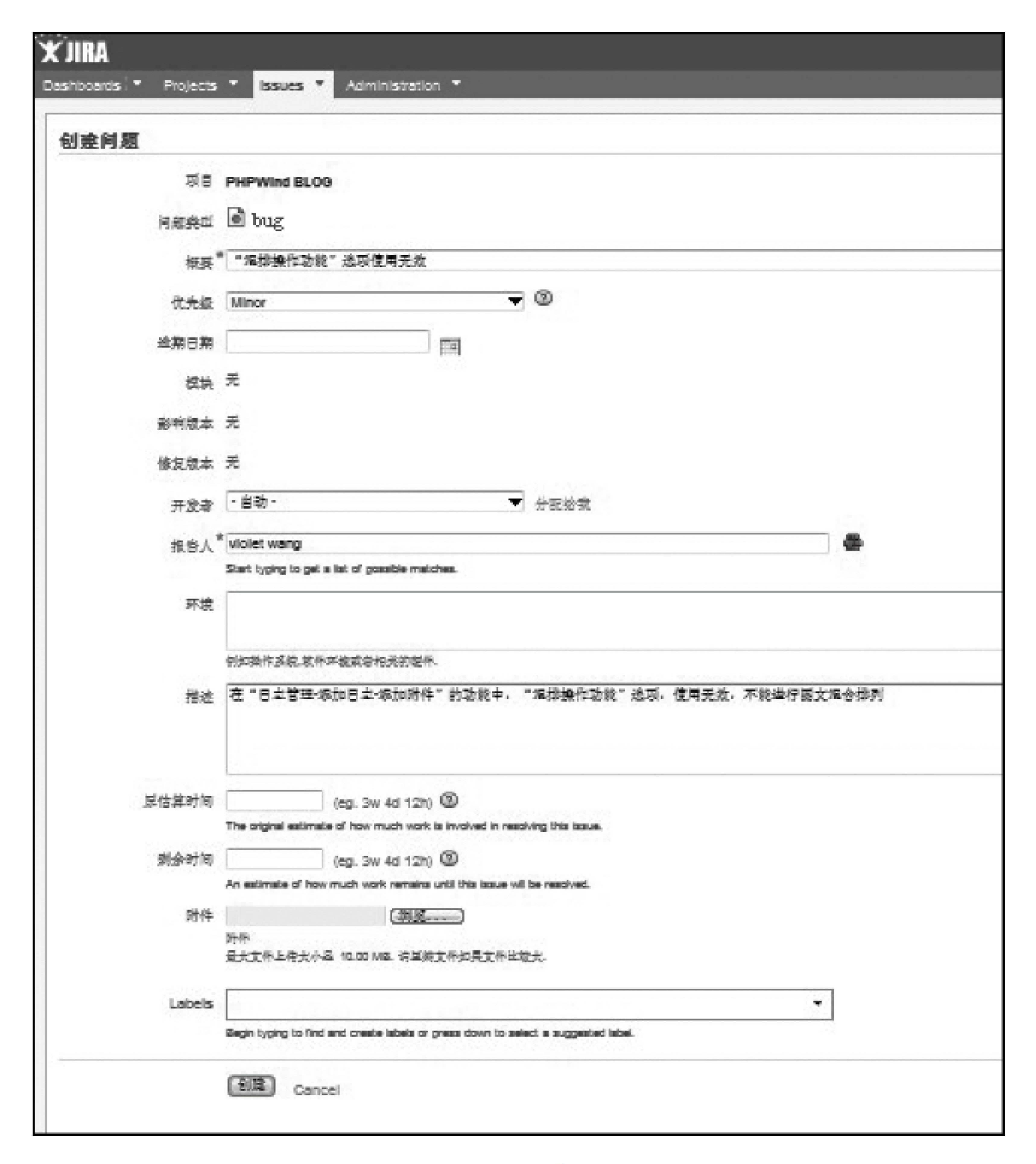


图 9-12　为 PHPWind BLOG 项目填写创建问题的详细信息界面

PHPWind BLOG 项目添加好的问题详细信息界面如图 9-13 所示。问题类型为 bug，具体描述为该 bug 的详细信息，优先级设置为 Minor。从图中可以看出目前对这个问题可以进行的操作有三个：开始进行、解决问题和 Workflow。问题创建后，可以

给问题上传附件，填写备注，复制和编辑问题。如果有权限，还可以删除问题，创建子任务等。



图 9-13　PHPWind BLOG 项目问题详细信息界面图

9.3.7　浏览项目

在浏览项目的界面上可以看到项目的基本信息资料，模块和版本信息及问题分布情况等，如图 9-14 所示。



图 9-14　浏览项目界面图

9.3.8　查找问题和配置过滤器

在查找问题界面上，可以在界面左侧进行查询条件的设置。JIRA 提供了强大的查询功能，在设定好查询条件后，可以将查询条件保存起来，定义为过滤器，供以后再次使用，甚至可以将保存的过滤器共享给团队的其他成员。

图 9-15 为"查找问题"的界面截图。

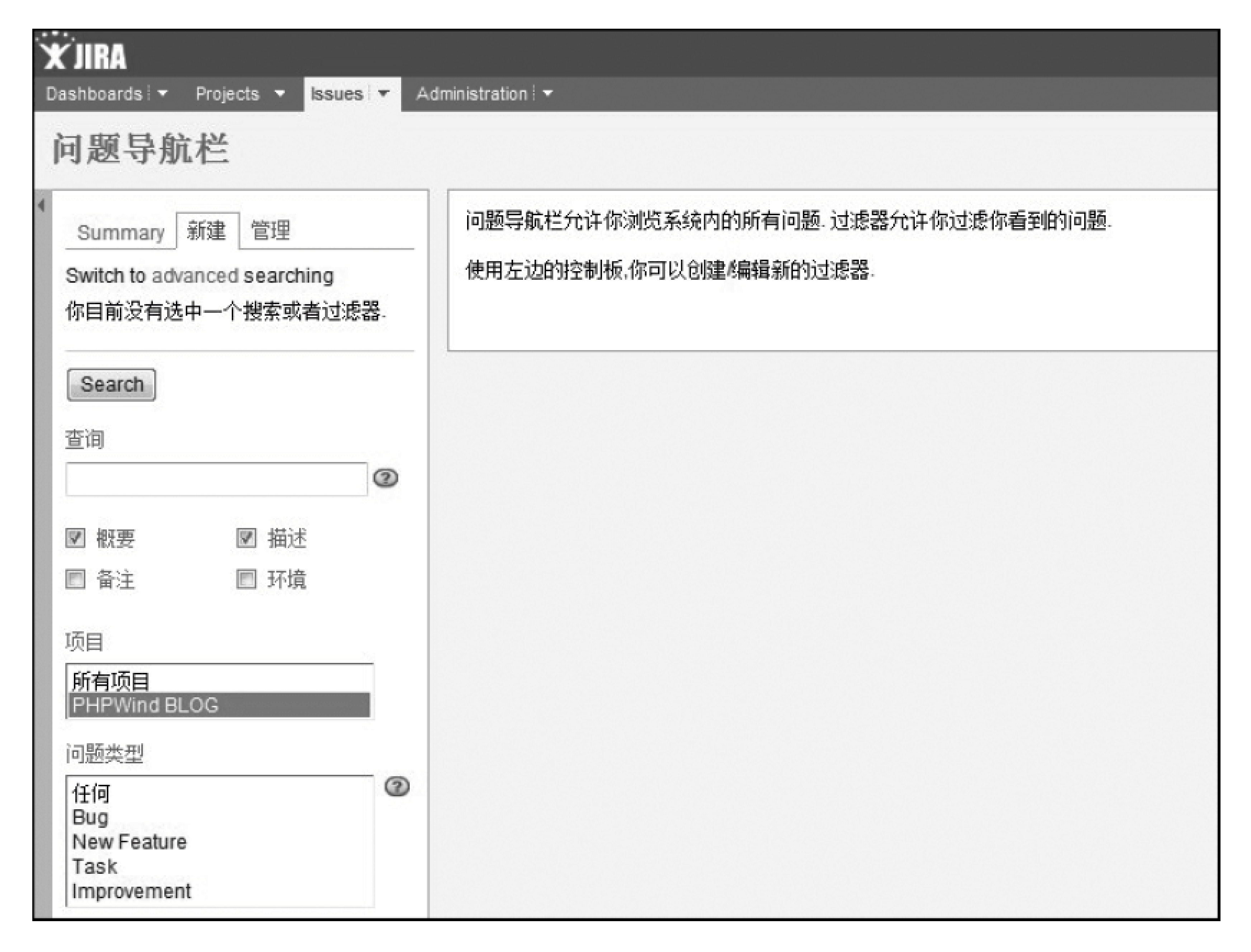


图 9-15　查找问题界面图

9.4 项目配置

9.4.1 添加项目和模块

项目和模块添加在前面已作描述,详见 9.3.3 节。

在图 9-6 所示界面上,输入项目名称、项目缩写、项目经理,选择邮件通知方案和权限设置方案即可完成项目添加。

在图 9-7 所示的项目详细信息界面上,单击 Components 下方的 Add,链接为项目创建组件/模块(Components)。PHPWind BLOG 项目添加组件的界面如图 9-16 所示。

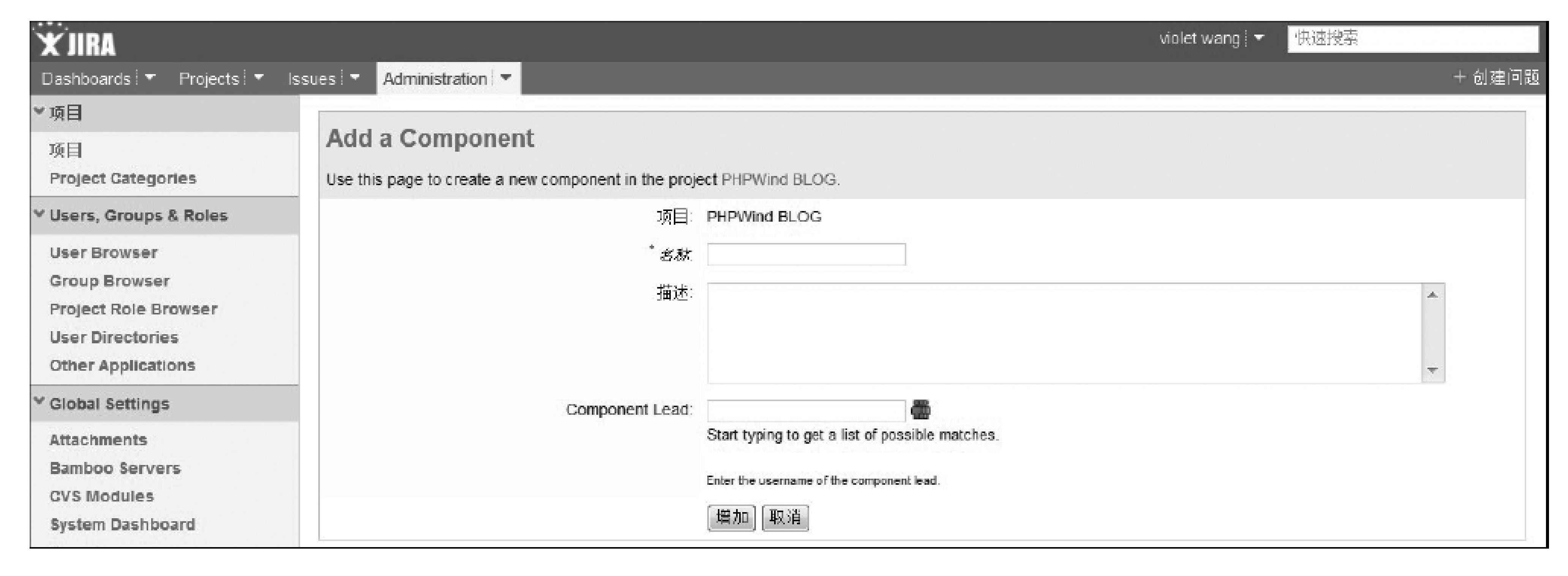


图 9-16　PHPWind BLOG 项目创建组件界面图

9.4.2 设置项目权限

项目权限的设置步骤如下：

① 单击 Administration→“项目”，单击项目名称链接查看项目详细信息界面。

② 在 Permission Scheme 部分，单击“选择”链接（见图 9-17），在进入的页面中选择需要的权限设置方案。



图 9-17 项目权限设置界面图

9.4.3 选择通知方案

设置步骤如下：

① 选择 Administration→“项目”，进入项目详细信息界面。

② 在 Notification Scheme 部分，单击 select scheme 链接，在进入的页面（见图 9-18）中选择需要的邮件通知方案。

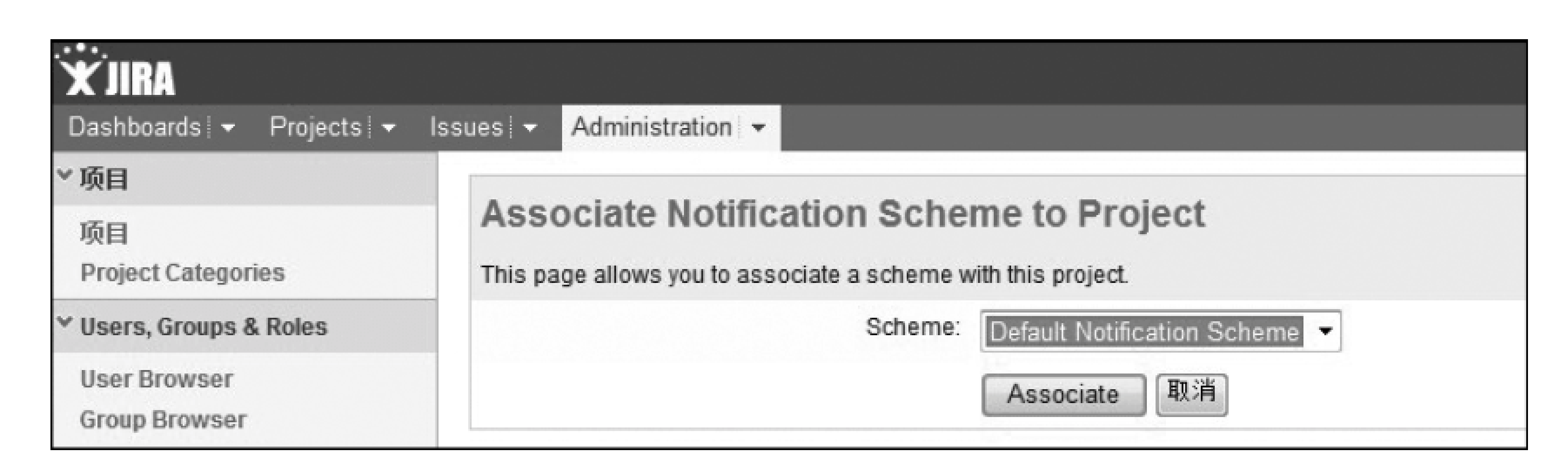


图 9-18 邮件通知方案设置界面

9.5 JIRA 系统的权限

JIRA 系统中的权限管理分为：系统级别、项目级别、问题级别以及注释级别。系统级是通过全局权限（Global Permissions）来管理的，影响 JIRA 中所有的项目和问题。针

对个别项目来说，可以通过 JIRA 系统中的权限模型(Permission Scheme)进一步配置项目级别的权限。下面将分别介绍全局权限设置步骤和基于项目的权限模型。

9.5.1 全局权限设置

设置步骤：

① 以 JIRA 系统管理员登录系统；

② 单击 Global Settings→Global Permissions。若要添加新的权限，可以在图 9-19 所示全局权限界面的下方单击“增加”按钮进行添加。

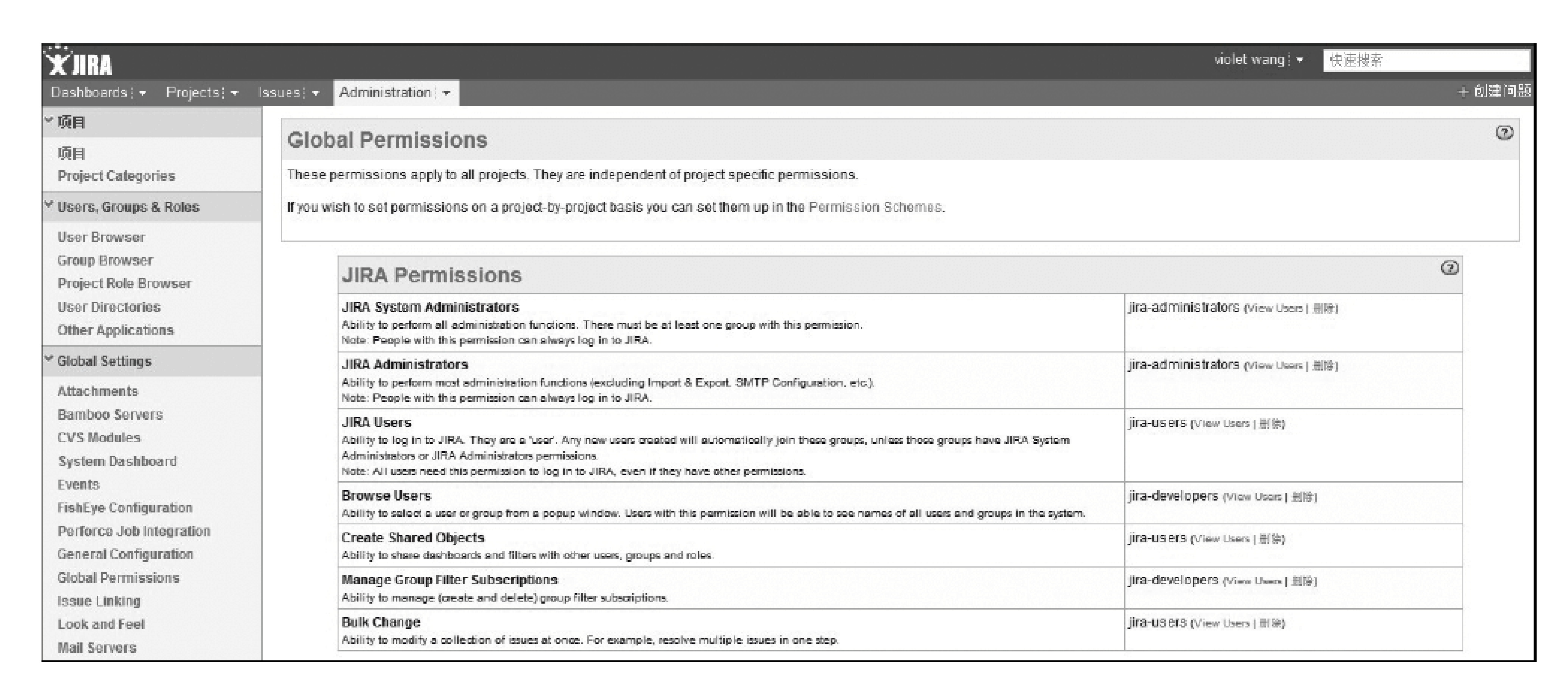


图 9-19 Global Permissions 界面

9.5.2 默认权限模型

JIRA 系统默认权限模型中的权限描述包含六部分：Project Permission(项目权限)、Issue Permission(问题权限)、Voters and Watchers Permission(投票者和关注者权限)、Comments Permission(评论权限)、Attachment Permission(附件权限)和 Time Tracking Permission(时间跟踪权限)。具体每个权限中的名称组成如表 9-3 所示。

表 9-3 JIRA 系统默认权限模型中的“项目权限”描述

权限名称	权限描述
Administer Projects	管理项目的权限，能够管理项目的组件/components 和版本/versions
Browse Projects	浏览项目；无此权限将无法浏览到 JIRA 系统中的项目；通常将此权限分配给项目组成员
Create Issues	创建问题(报告 bug)；通常将此权限分配给测试人员。
Edit Issues	编辑问题；拥有此权限可以对创建后的问题进行修改。通常将此权限分配给问题报告者、项目管理人员
Schedule Issues	设置或者编辑问题的预期完成日期；通常将此权限分配给问题报告者、项目管理人员

续表

权 限 名 称	权 限 描 述
Move Issues	在项目之间移动问题;只能移动到有创建问题权限的项目上;通常将此权限分配给项目管理人员
Assign Issues	分配问题;通常将此权限分配给测试人员、项目管理人员
Assignable User	可以分配到问题的人;通常将此权限分配给开发人员
Resolve Issues	解决和重新打开问题,可以设置修复版本;通常将此权限分配给开发人员
Close Issues	关闭问题;通常将此权限分配给问题报告人员或 QA 人员
Modify Reporter	创建或编辑问题时修改报告者;通常将此权限分配给问题报告人员或项目管理人员
Add Comments	添加注释;通常将此权限分配给项目组所有人员
Delete All Comments	删除所有注释;通常将此权限分配给项目管理人员
Delete Own Comments	删除自身的注释;通常将此权限分配给项目组所有人员
Edit All Comments	编辑所有注释;通常将此权限分配给项目开发人员
Edit own Comments	编辑自身注释;通常将此权限分配给项目组成员
Delete Issues	删除问题、注释和附件;通常将此权限分配给问题报告人员或项目管理人员
Work On Issues	针对问题完成情况记录,需启用 Time Tracking;通常将此权限分配给开发人员
Link Issues	将相关问题链接到一起,需启用 Issue Linking;通常将此权限分配给问题报告人员或项目管理人员
Create Attachments	添加附件;通常将此权限分配给问题报告人员、开发人员和项目管理人员
View Version Control	查看问题提交版本控制信息;将此权限分配给问题报告人员、开发人员和项目管理人员
View Voters and Watchers	查看投票者和关注者列表信息;通常将此权限分配给 JIRA 系统管理人员
Set Issue Security	设置问题的安全级别,只有处于该安全级别的用户才可以看到问题。通常将此权限分配给 JIRA 系统管理人员

第 10 章



软件质量保证与软件测试

有一则故事流传很广。魏文王问扁鹊："你们家兄弟三人都精于医术，到底哪一位最好呢?"扁鹊答："长兄最好，中兄次之，我最差。"文王又问："那为什么你却最出名呢?"扁鹊答："长兄治病于病情发作之前，诊疗前后无甚感觉，一般人不知他事先已除病因，所以名气全无；中兄治病于病情初起之时，一般人以为他只能治小病，所以他的名气只及本乡；而我治病于严重时，人们总看到我在经脉上穿针放血，在皮肤上开刀敷药，以为我的医术最高，因此名气响遍全国。"理性告诉我们：事后控制不如事中控制，事中控制不如事前控制，这就是所谓的"防患于未然"。

10.1 质量保证

软件测试是使用人工或者自动手段来运行或测试某个系统的过程，其目的在于检验软件是否满足规定的需求或弄清预期结果与实际结果之间的差别，核心目的是为了确保开发的产品适合需求、具备良好的质量。这其实是属于质量管理(Quality Management，QM)工作的范畴，而质量管理也同样遵循"防患于未然"这条原则。

10.1.1 全面质量管理

在介绍全面质量管理之前，我们首先明确一下有关质量的定义。国家标准(GB/T19000-2008，ISO9000：2005)对质量下的定义为：一组固有特性满足要求的程度。目前更流行、更通俗的定义是从用户的角度去定义质量：质量是用户对一个产品(包括相关的服务)满足程度的度量。

在提出全面质量管理概念之前，质量管理经历了两个发展阶段：

(1) 20 世纪 30 年代以前为质量检验阶段，仅能对产品的质量实行事后把关。但质量并不是检验出来的，所以，质量检验并不能提高产品质量，只能剔除次品和废品。

(2) 1924 年休哈特提出，产品质量不是检验出来的，而是生产制造出来的，质量控制的重点应放在制造阶段，从而将质量控制从事后把关提前到制造阶段，质量控制从检验阶段发展到统计过程控制阶段，利用休哈特工序质量控制图进行质量控制。

1961 年美国的菲根堡姆(Armand Vallin Feigenbaum)提出了全面质量管理理论(Total Quality Management,TQM)[37-38],将质量控制扩展到产品寿命循环的全过程,强调全体员工都参与质量控制,其含义远远超出了一般意义上的质量管理的领域,而成为一种综合的、全面的经营管理方式和理念。

全面质量管理是企业管理现代化、科学化的一项重要内容。它后来在西欧与日本逐渐得到推广与发展。它应用数理统计方法进行质量控制,使质量管理实现定量化,变产品质量的事后检验为生产过程中的质量控制。全面质量管理类似于日本式的全面质量控制(Total Quality Control,TQC):首先,质量的含义是全面的,不仅包括产品服务质量,而且包括工作质量,用工作质量保证产品或服务质量;其次,是全过程的质量管理,不仅要管理生产制造过程,而且要管理采购、设计直至储存、销售、售后服务的全过程。

全面质量管理要求必须形成一种这样的意识,好的质量是设计、制造出来的,是由过程所决定的,而不是检验出来的;质量管理的实施要求全员参与,并且要以数据为客观依据,要视顾客为上帝,以顾客需求为核心;在实现方法上,要一切按 PDCA 循环办事。

PDCA 循环(PDCA Cycle)的概念最早是由美国质量管理专家戴明(W. E. Deming)于 20 世纪 50 年代初提出的,所以又称为“戴明环”,如图 10-1 所示。它是全面质量管理所应遵循的科学程序,全面质量管理活动的全部过程。PDCA 循环是能使任何一项活动有效进行的一种合乎逻辑的工作程序,特别是在质量管理中得到了广泛的应用。P、D、C、A 四个英文字母所代表的意义如下:

(1) P(Plan)——计划。包括方针和目标的确定以及活动计划的制定。

(2) D(Do)——执行。执行就是具体运作,实现计划中的内容。

(3) C(Check)——检查。就是要总结执行计划的结果,分清哪些对了、哪些错了,明确效果,找出问题 。

(4) A(Action)——行动(或处理)。

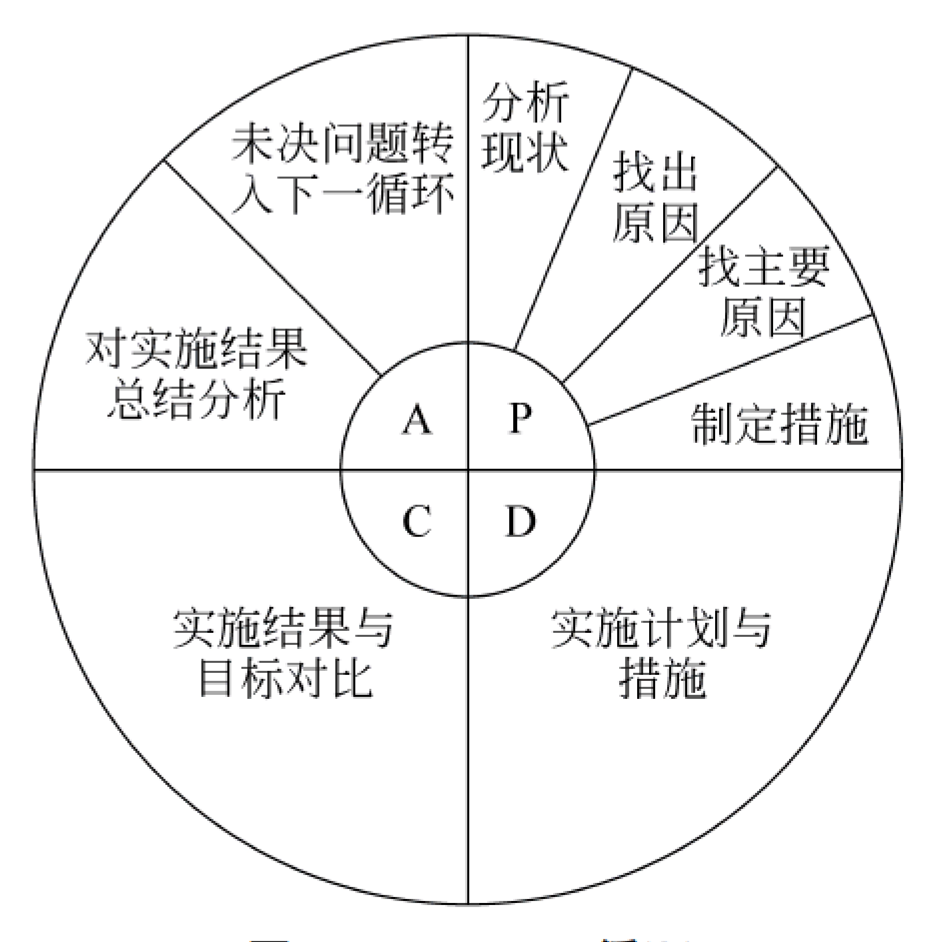


图 10-1 PDCA 循环

对总结检查的结果进行处理,成功的经验加以肯定,并予以标准化,或制定作业指导书,便于以后工作时遵循;对于失败的教训也要总结,以免重现。对于没有解决的问题,应提给下一个 PDCA 循环中去解决。

PDCA 循环有以下四个明显特点:

(1) 周而复始:PDCA 循环的四个过程不是运行一次就完结,而是周而复始地进行。一个循环结束了,解决了一部分问题,可能还有问题没有解决,或者又出现了新的问题,再进行下一个 PDCA 循环,依此类推。

(2) 大环带小环:类似行星轮系,一个公司或组织的整体运行体系与其内部各子体系的关系,是大环带动小环的有机逻辑组合体。

(3) 阶梯式上升:PDCA 循环不是停留在一个水平上的循环,不断解决问题的过程就是水平逐步上升的过程。

(4) 统计的工具：PDCA 循环应用了科学的统计观念和处理方法。作为推动工作、发现问题和解决问题的有效工具。

10.1.2 质量保证与质量控制

在实际质量工作中，经常会碰到一个问题，即质量保证(Quality Assurance，QA)和质量控制(Quality Control，QC)关系比较容易混淆。QA 和 QC 由于都属于质量管理的范畴，同时二者的工作存在部分交叉、共同的工作内容，因此容易造成概念和职责上的混淆。按照 ISO 9000:2000，QA 的定义是“质量管理的一部分，致力于提供质量要求会得到满足的信任”，QC 的定义是“质量管理的一部分，致力于满足质量要求”。具体地说，QA 是对人、对过程，致力于使管理者、顾客和其他相关方相信有能力满足质量要求，更多的是体现在流程制度上；QC 是对人、对事与对物，直接致力于满足质量要求，体现在质量把关的具体工作过程中。

QA 和 QC 各司其职，相辅相成，统一于质量管理(QM)。Crosby[39]曾经打过一个比喻：就像一部汽车，质量控制(QC)就是所有那些告诉你汽车当前运动状态的仪器仪表；质量保证(QA)包括各类标准，是告诉你所有部件操作方法的用户手册；而质量管理(QM)则是你要追求的目标，比如希望能平安、高速地驾驶汽车。可以看出，为了实现质量管理的目标，质量保证和质量控制都是不可或缺的部分。

软件信息化方面的一些标准对 QA 进行了多种定义。计算机软件质量保证计划规范 GB/T 12504-1990 指出，质量保证是指为使软件产品符合规定需求所进行的一系列有计划的必要工作；软件工程术语 GB/T11457-1995 指出，质量保证是为使某项目或产品符合已建立的技术需求提供足够的置信度，而必须采取的有计划和有系统的全部动作的模式。这两个标准中都没有直接关于 QC 的定义。按照不同的目的、从不同的角度对同一个术语的定义往往存在差异，例如 GB/T 12504-1990、GB/T11457-1995 分别对 QA 的定义就存在差异，按照 GB/T 12504-1990 的 QA 定义涵盖的范围较宽，包含了 QC 的内容。

QA 和 QC 的具体区别，如表 10-1 所示。

表 10-1　QA 和 QC 的区别

	QA	QC
全称	质量保证(Quality Assurance，QA)	质量控制(Quality Control，QC)
角色	QA 工程师	测试工程师，评审员
定义	为了确保软件开发过程、工件符合预期的结果，依照过程和计划采取的一系列活动及其结果评价	为了发现软件产品的错误、缺陷而进行工作的过程
职责	监控公司质量保证体系的运行情况，审计项目的实际执行情况和公司规范之间的差距，并出具统计分析报告和改进建议。是过程、产品的审计者	对每个阶段或者关键点的工件进行检查，评估工件是否符合预计的质量要求。关注各阶段的评审和测试缺陷。是产品质量检查者
分工原则	QA 只要检查项目按照过程进行了某项活动没有，产出了某个产品没有	QC 检查产品是否符合质量要求

QA与QC的其他重大区别还包括：具备必要资质的QA工程师是组织中的高级人才，需要全面掌握组织的过程定义，熟悉所参与项目所用的工程技术；QC工程师则既包括软件测试设计员等高级人才，也包括一般的测试员等中、初级人才。国外软件企业要求QA工程师应具备两年以上的软件开发经验，半年以上的分析员、设计员经验；不仅要接受QA工程师方面的培训，还要接受履行项目经理职责方面的培训。在项目组中，QA工程师独立于项目经理，不由项目经理进行绩效考核；QC工程师受项目经理领导，通常在项目运行周期内QC工程师的绩效大部分由项目经理考核决定。

虽然不同公司有不同的情况，但是原则都是一样的。QA工程师是从过程和标准来控制开发过程，从而达到提高软件质量的目的。而QC工程师则是通过测试、评审等验证、确认手段来发现软件中的缺陷，并确保该缺陷得到解决，从而达到提高软件质量的目的。

如果企业QA人员配备不足，可以先确定由QC兼任QA工作。但是只能是暂时的，独立的QA人员应当具备，因为QC工作也是要遵循过程要求的，也是要被审计过程的，这种混合情况，难以保证QC工作的过程质量。但是在软件企业规模较小的情况下，所有测试、评审、审计等质量管理相关的工作都可以统称为QA。

10.1.3 软件质量标准

行业标准是由一些行业机构、学术团体和国防机构制定，并适用于某个业务领域的标准。经过几十年的发展，软件行业发展出了几个重要的、应用面广的行业标准。

ISO 9000系列国际标准是在总结了英国的国家标准基础之上产生的，国际标准化(英文简称ISO)于1987年发布了ISO 9000《质量管理和质量保证标准——选择和使用指南》、ISO 9001《质量体系——设计开发、生产、安装和服务的质量保证模式》等6项国际标准，统称为ISO 9000系列标准。

目前，ISO 9000系列管理标准已经为提供产品和服务的各行各业所接纳和认可，ISO 9000系列质量体系被世界上110多个国家所广泛采用，使市场竞争更加激烈，产品和服务质量得到日益提高。软件企业贯彻实施ISO 9000质量管理体系认证，应当选择质量保证模式标准ISO 9001。ISO 9000-3作为软件企业实施ISO 9001质量保证模式标准的实施指南。通过对软件产品从市场调查、需求分析、软件设计、编码、测试等开发工作，直至作为商品软件销售，以及安装及维护整个过程进行控制，保障软件产品的质量。现在ISO 9000标准已被各国软件企业广泛采用，并将其作为建立企业质量体系的依据。

为支持美国国防部对软件承包商的能力进行客观评价，卡内基·梅隆大学软件工程研究所(SEI)于1991年发表了《软件能力成熟度模型》即SW-CMM 1.0版，1999年推出了SW-CMM的修订本《系统工程和软件工程综合能力成熟度模型》即CMMI-SE/SW 1.0版。

经过一段时间等级评估的运行，CMM评估对软件工程改进产生了明显的促进作用，自1990年以来，SEI把基于CMM的评估作为商业行为推向市场。多年来，接受CMM评估的软件企业已扩展到很多国家与地区。

ISO 9000和CMM都是以全面质量管理为理论基础，ISO 9000是适用于所有专业领

域的一种质量保证标准,CMM 则是专门针对软件行业的描述软件过程能力专用模型。ISO 9000 强调以顾客满意为目标,而 CMM 则着重于评价承包商的软件成熟能力。

10.2 软件质量保证

ISO/IEC 12207:1995 指出,质量保证是一个有计划、有组织的活动,它向所有相关的人提供证据,以证明质量功能正在按质量要求运行的信心。而软件质量保证过程(Software Quality Assurance, SQA)是恰当保证为项目生存周期中的软件产品和过程符合规定需求和已制订计划提供足够保证的过程。

如果将一个软件生产类比于一个工厂的生产。那么生产线就是过程,产品按照生产线的规定过程进行生产。软件质量保证的职责就是保证过程的执行,也就是保证生产线的正常执行。具体来说,SQA 是通过有计划地进行各种评审和审核来验证和确认生产出来的软件过程、产品是否合乎标准的系统工程,是建立一套有计划、系统的方法,来向管理层保证拟定出的标准、步骤、实践和方法能够正确地被所有项目所采用。

SQA 的目的是使软件过程对于管理人员来说是可见的。它通过对软件过程、产品进行评审和审计来验证软件是合乎标准的。一般认为,SQA 包含:

1) 一种质量管理方法;
2) 有效的软件工程技术、方法和工具;
3) 在整个软件工程过程中采用正式的技术评审;
4) 一种多层次的测试策略;
5) 对软件代码、文档和数据及其修改的控制;
6) 保证软件开发适时遵循开发标准;
7) 度量和报告机制[40]。

10.2.1 SQA 概要

SQA 的职责可以总结为如下,主要包括:

(1) 负责质量保证体系的实施与维护,制订项目的 SQA 计划,定义项目质量策略、度量过程和产品质量状态,指导开发团队达到质量目标。

(2) 协助项目组进行规范化的项目管理,进行流程培训和咨询活动。

(3) 负责监控、稽核流程在研发团队的实施情况。

(4) 实施 SQA 评审和组织评审活动,对项目开发管理活动及产品进行审查或审计,对风险和质量问题进行跟踪直到解决或关闭。

(5) 定期和不定期地进行项目质量报告,向高层领导报告公司软件项目实施过程中未能及时处理的不合格项,以及相关风险。

(6) 对开发流程的效率和质量、绩效指标进行收集和分析,发现流程中的问题并能够持续改进软件过程。

根据文献[41],软件质量保证活动,可以用图 10-2 描述。

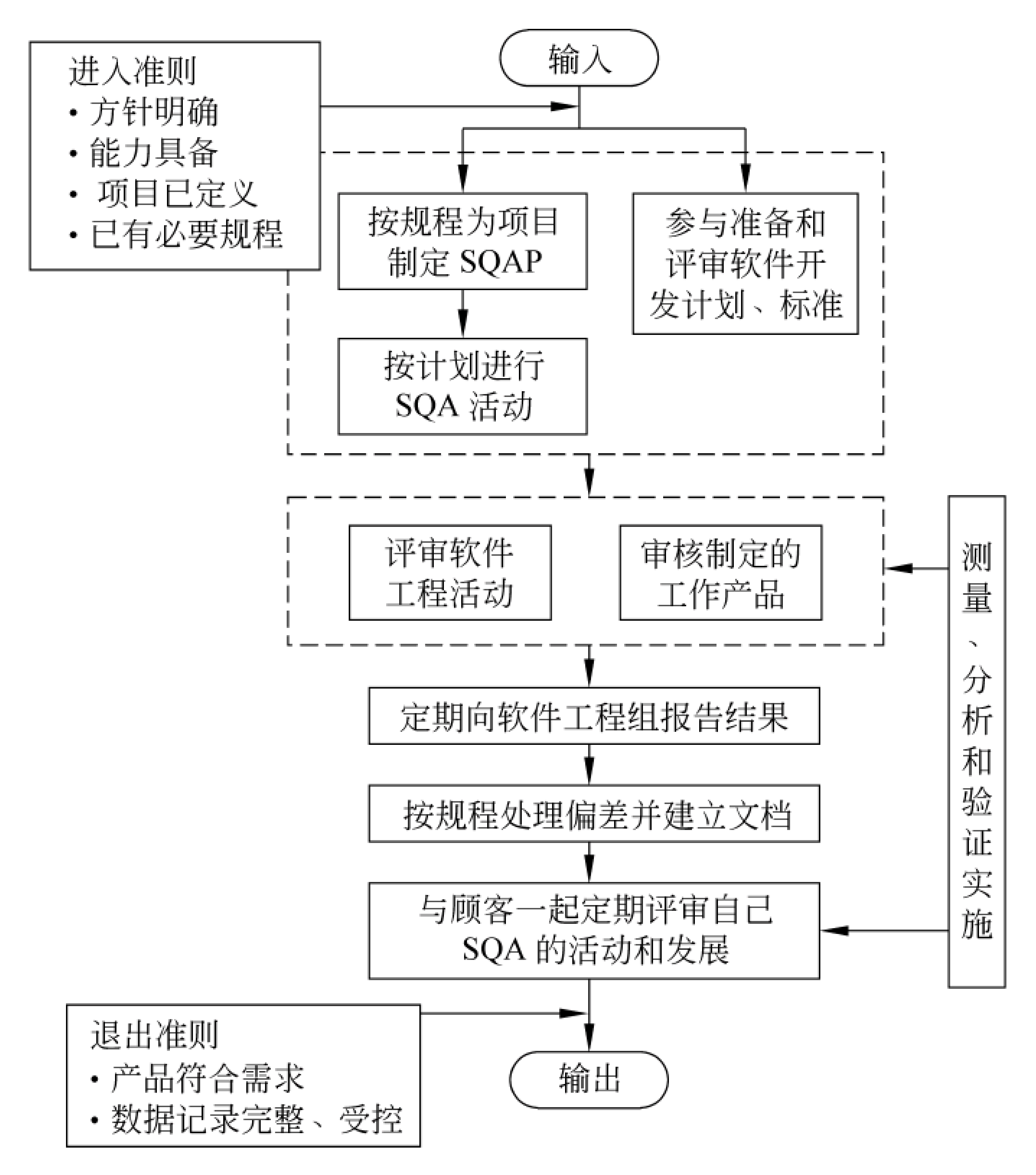


图 10-2 软件质量保证活动概要

10.2.2 SQA实施的步骤、措施

SW-CMM 对 SQA 的活动描述只是给出了要求，没有涉及具体措施。为了实施软件质量保证活动，胡铮[41]给出了一个基于全面质量管理思想 PDCA 质量循环的质量保证活动实施的步骤，如图 10-3 所示。

(1) 目标(target)：设定质量特性与子特性的评价标准。软件质量保证小组在新项目的计划阶段就应开始介入，对用户提出的使用质量需求进行分析，将这些使用质量需求的分解、评价与软件的外部质量需求对应起来，确定软件产品应满足的外部质量特性，诸如功能性、可靠性、易用性、效率等。然后再对它们进行分析，通过质量展开(分析)，确定软件开发必须满足的内部质量特性，及其度量或评价的标准(用一系列指标值表示)，形成正式文档。

(2) 计划(plan)：与组织的软件过程组(SEPG)和项目管理人员合作，参与制定项目的软件过程，包括如何划分开发阶段、分解任务、定义过程活动(人员、任务、中间产品、工作流)。确定适合于被开发软件各个阶段、各个活动的质量评测检查项目与质量度量方法，包括产品质量和过程质量。考虑到评价工作量，检查项目数一般设定为 20～30 个。与此同时还要研讨实现质量目标的方法和手段。

(3) 实施(do)：在软件开发的过程中，参与各个活动的评审和阶段的正式技术评审。对软件开发各个阶段的进展、完成质量及出现的问题进行监督，审核各项活动和中间产

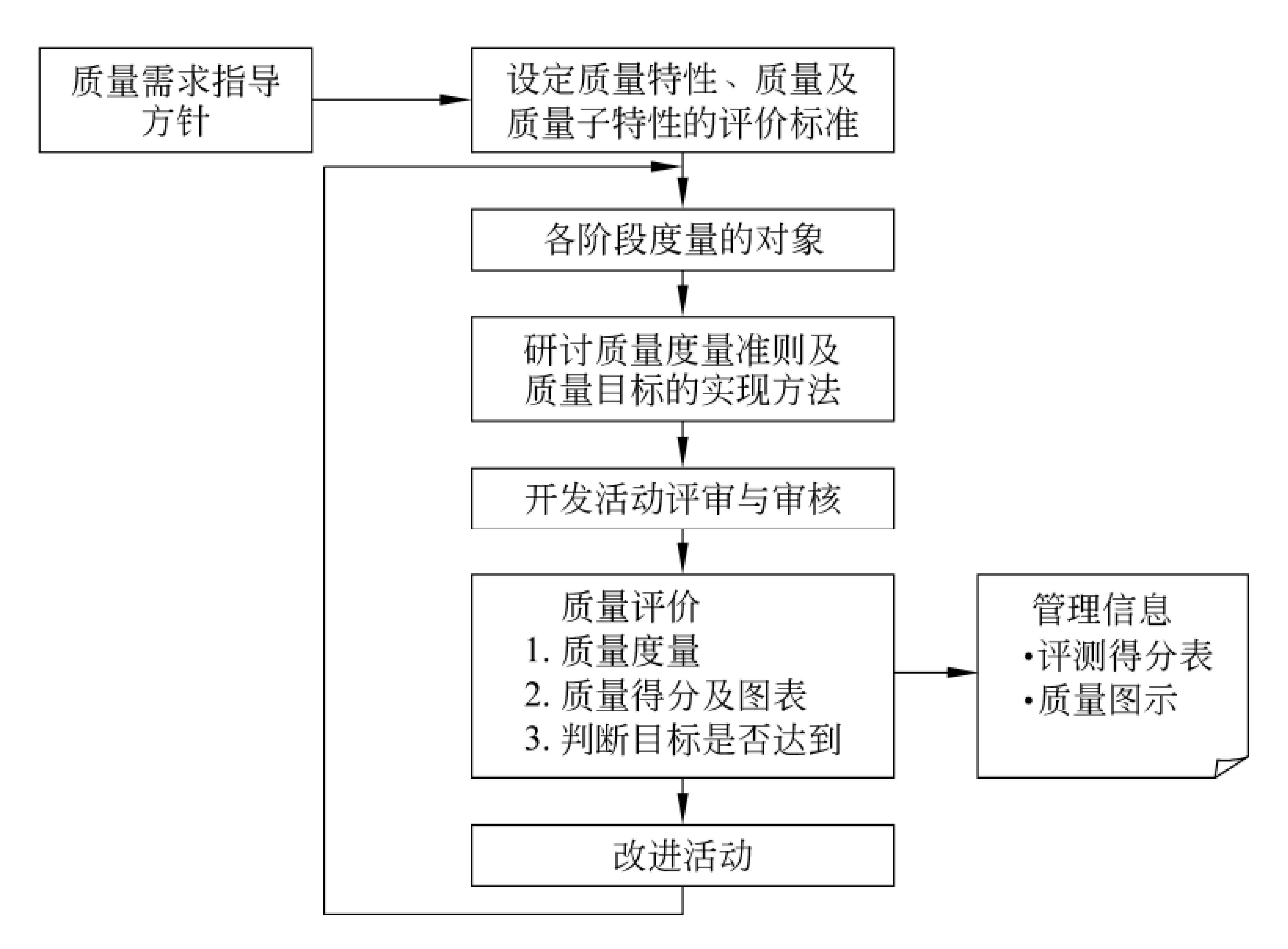


图 10-3　软件质量保证实施的步骤

品的生成是否遵守相应的过程规范，形成报告。如果发现偏差或不符合问题，遵循逐级解决的原则进行解决，将处理结果通知所有相关人员，记录解决的过程及结果，以作为日后改进的重要参考资料。

(4) 检查(check)：根据计划阶段确定的质量度量标准进行评价，算出得分，比较评价结果的质量得分和质量目标，看其是否合格。

(5) 行动(action)：对评价发现的问题进行改进活动，如果实现并达到了质量目标就转入下一个开发阶段。这样重复计划到行动的过程，直到整个开发项目完成。

实际上，SQA 工作的核心内容在于审计，以上活动都是围绕审计进行的计划、证实和跟踪工作，从审计角度出发，可以把 SQA 工作分为三块：

1. 计划

针对具体项目制定 SQA 计划，确保项目组正确执行过程。制定 SQA 计划应当注意如下几点：

- 有重点：依据企业目标以及项目情况确定审计的重点。
- 明确审计内容：明确审计哪些活动，哪些产品。
- 明确审计方式：确定怎样进行审计。
- 明确审计结果报告的规则：审计的结果报告给谁。

2. 审计/证实

依据 SQA 计划进行 SQA 审计工作，按照规则发布审计结果报告。注意审计一定要有项目组人员陪同，不能搞突然袭击。双方要开诚布公，坦诚相对。

审计的内容：是否按照过程要求执行了相应活动，是否按照过程要求产生了相应产品。

3. 问题跟踪

对审计中发现的问题，要求项目组改进，并跟进直到解决。

10.2.3 SQA活动

SQA 与做技术工作的软件工程师和负责质量保证的计划、监督、记录、分析及报告工作的 SQA 小组相关。

软件工程师通过采用可靠的技术方法和措施，进行正式的技术评审，执行计划周密的软件测试来考虑质量问题，并完成软件质量保证和质量控制活动。SQA 小组的职责是辅助软件工程小组得到高质量的最终产品。

SQA 小组完成以下任务。

（1）为项目准备 SQA 计划。

该计划在制定项目时确定，由所有感兴趣的相关部门评审。计划包括：

- 需要进行的审计和评审。
- 项目可采用的标准。
- 错误报告和跟踪的规程。
- 由 SQA 小组产生的文档。
- 向软件项目组提供的反馈数量。

（2）参与开发项目的软件过程描述。

评审过程描述以保证该过程与组织政策，内部软件标准，外界标准以及项目计划的其他部分相符。

（3）评审各项软件工程活动，对其是否符合定义好的软件过程进行核实。

记录、跟踪与过程的偏差。

（4）审计指定的软件工作产品，对其是否符合事先定义好的需求进行核实。

对产品进行评审，识别、记录和跟踪出现的偏差；对是否已经改正进行核实；定期将工作结果向项目管理者报告。

（5）确保软件工作及产品中的偏差已记录在案，并根据预定的规程进行处理。

（6）记录所有不符合的部分并报告给高级领导者。

目前，实施 CMM 的企业越来越多。CMM 模型就要求建立 QA 角色，一般还要求 QA 独立于项目组。从国内来看，多数的 QA 没有技术背景，检查不出有意义的偏差，而且缺乏领导支持，做起来十分困难。

缺乏信任和支持只是一个方面，QA 工作本身就很具挑战性。它要求 QA 具有软件工程的知识、软件开发的知识、行业背景的知识、数理统计的知识、项目管理的知识、质量管理的知识等。需要指出的是，SQA 应该具备以下素质，才能确保质量保证活动的顺利开展。

- 过程为中心：应站在过程的角度考虑问题，只要保证了过程，QA 就尽到了责任。
- 服务精神：为项目组服务，帮助项目组确保正确执行过程 。
- 了解过程：深刻了解企业的工程，并具有一定的过程管理理论知识 。

• 了解开发：对开发工作的基本情况了解，能够理解项目的活动。
• 善于沟通：能够营造良好的气氛，避免审计活动成为一种找茬活动，艺术化地处理人际及工作关系。

10.3 评　　审

在软件开发过程(尤其是分析和设计阶段)中，为检测出任何可能已经引入的错误，系统分析员或编制文档的分析员必须反复检查设计文档。此外，开发组长也必须考察这份文档及其细节，以在批准它之前检测出任何残留的错误。然而，显然由于这些专业人员参与这份文档的生成，他们不大可能检测出他们自己的错误，即使进行多次检查也是如此。所以，只有别的人——例如同事、上级、专家和顾客代表(他们有不同的经历和观点，也没有直接参与这份文档的建立)——才有能力评审这个产品，并检测出未被开发组注意到的错误。

没有经过他人评审，会造成缺陷不断放大。其模型[42]如图 10-4 所示。



图 10-4　缺陷放大模型

图 10-4 表示某一个开发阶段经过错误的放大、新增和检测过滤等步骤后，缺陷的数目变化。来自上一步骤的错误会分为两个部分，一个部分会原样保留；另一个部分会被放大，导致更多的错误，再加上新产生的错误，这构成了某个阶段的所有错误数；再经过检测过滤掉一部分后，就是传给下一个步骤的错误数。

图 10-5 演示了一个典型的无评审环节的缺陷放大情况，最终集成测试阶段会剩余 94 个错误。相比之下，对于概要设计和详细设计阶段引入评审过滤掉一些缺陷后，图 10-6 集成测试阶段仅剩余 24 个错误，大大减少的后续测试的错误数目。

10.3.1 评审概要

IEEE(1990)定义评审过程是：“一个过程或一次会议，通过它，一个工作产品或一组工作产品被提交给项目人员、经理、用户、顾客或其他感兴趣者，以征求意见或得到批准。”由于这些文档是项目初始阶段的产品，评审在 SQA 过程中具有特殊的重要性，因为它们提供早期检测并防止将设计和分析错误传给“下游”。在下游阶段，错误检测和改正要复杂得多、麻烦得多，因而费用也高得多。

评审一般有正式技术复审和同行评审(包括审查和走查)两种形式。应当注意，审查

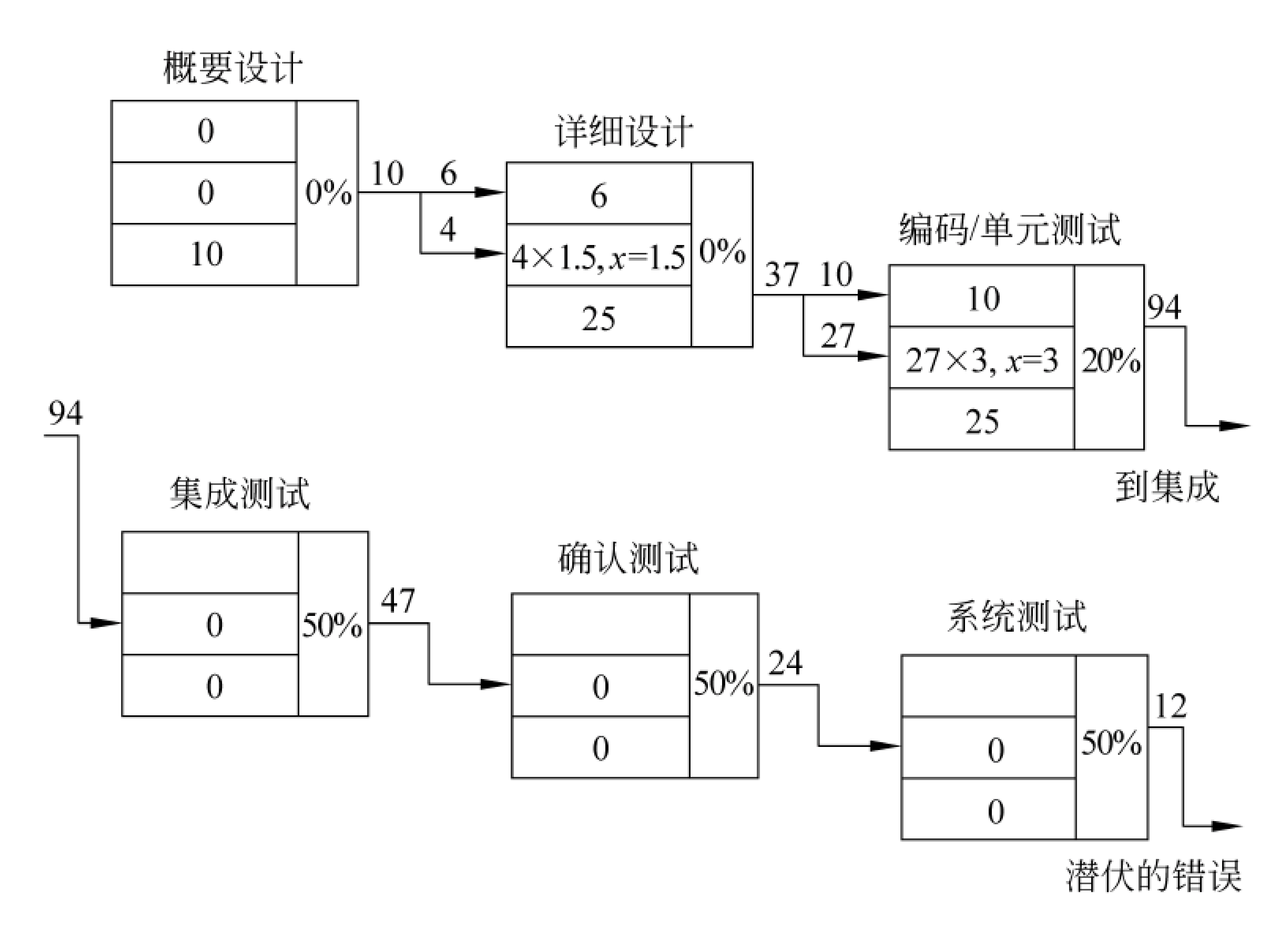


图 10-5　无评审的缺陷放大情况

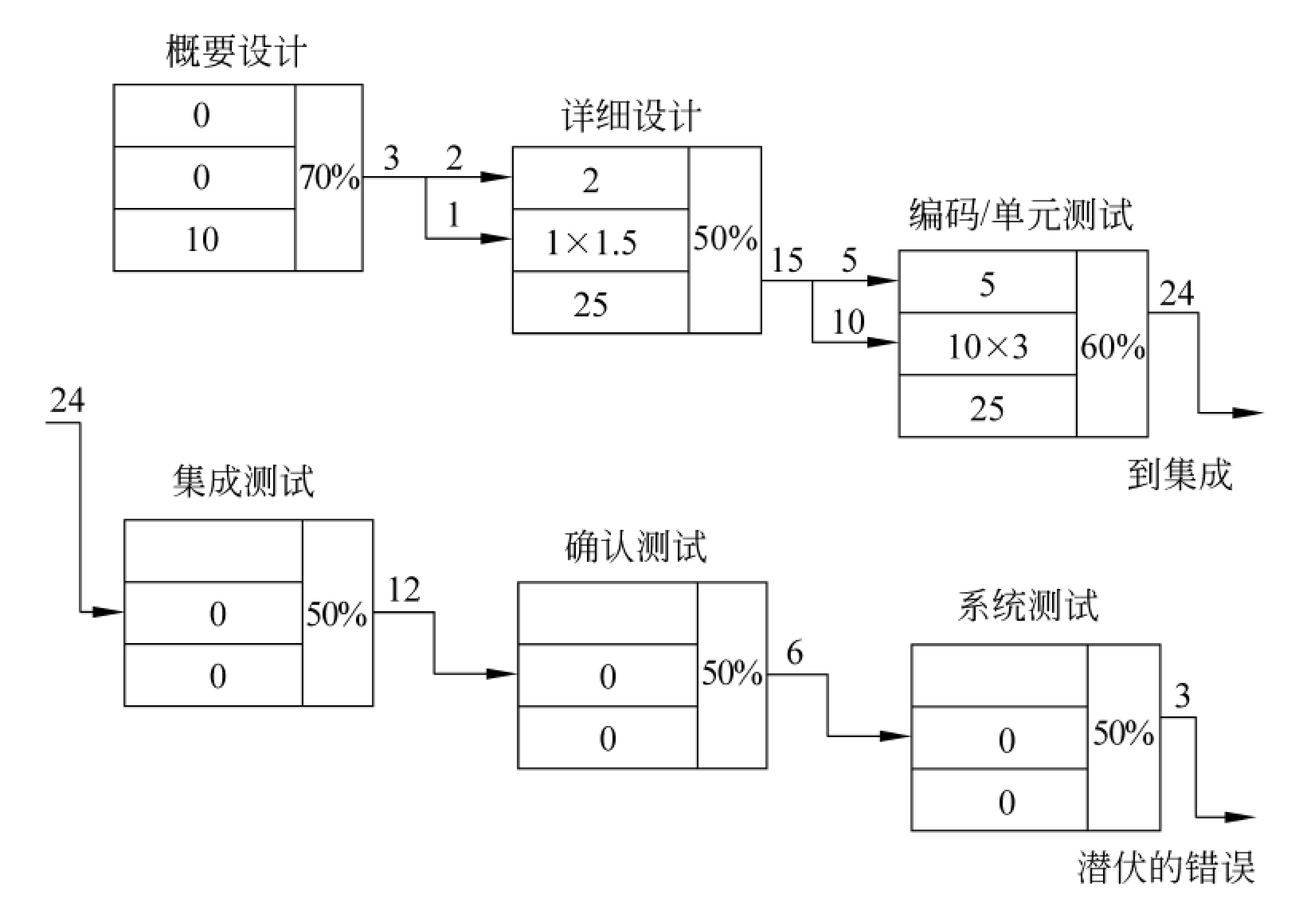


图 10-6　有评审的缺陷放大情况

和走查的成功执行也检测编码阶段的缺陷，这时评审的合适文档是打印的代码。

有几个目标驱动评审。评审的直接目标同当前项目有关，而其间接目标有更普遍的性质，同评审本身对提升开发组成员的专业知识和改进机构所用开发方法学的特有贡献有关。

以下是评审的主要目标：

- 检测需求分析和设计错误，以及关于初始规格书和批准的更改需要进行改正、更改和完善的主题。
- 确定可能影响项目完成的新风险。
- 找出偏离模板和风格规程及约定的地方。改正这些偏离预计会带来方法和文档

风格的一致性，这必然对改进交流和协作做出贡献。

- 批准分析或设计产品，使项目继续前进到下一开发阶段。

间接目标：

- 提供一个非正式会议场所，以交换关于开发方法、工具和技术方面的专业知识。
- 记录分析和设计错误，这些错误将用做未来改正性措施的基础。预期改正性措施将通过提高有效性和质量以及其他产品特性来改进开发方法。

各种评审方法的不同在于强调不同的目标和成功实现每个目标的程度。所以，为了更好地“过滤”掉错误和更大的长期影响，应当使用双层甚至三层“过滤网”，这种网是根据可用的评审方法的范围构造出来的。

评审不是随意进行的活动，程序性的制度和协同工作处于正式技术复审、审查和走查的核心，要求每一位参加者在给出意见时突出其责任或专长。在每个评审会议上，分配人员记录互相同意的意见，后续的事项清单应当包括缺陷位置和描述的全部细节，且允许开发组后续进行全面检索。然而，因为人们有试图现场设计解决方案的嗜好，经常离开正在讨论的问题，甚至在会议期间去处理个人事务，所以须有一个主持人，控制讨论正常进行。

一般知道要对分析或设计的产品进行评审，会激励开发组工作得更好。这代表了评审对改进产品质量的另一个贡献。

10.3.2 正式技术复审

正式技术复审（Formal Technical Review，FTR），也称之为设计评审（Development Review，DR），同其他评审的不同在于，它是批准设计产品所必需的唯一评审。不经过批准，开发组就不能继续到软件开发项目的下一阶段。正式技术复审可以在需要完成一份分析或设计文档的任何开发里程碑进行，不论该设计文档是一份需求规格书还是安装计划。以下给出了常见的正式技术复审的清单。

- DPR——开发计划评审（Development Plan Review）；
- SRSR——软件需求规格书评审（Software Requirement Specification Review）；
- PDR——概要设计评审（Preliminary Design Review）；
- DDR——详细设计评审（Detailed Design Review）；
- DBDR——数据库设计评审（DataBase Design Review）；
- TPR——测试计划评审（Test Plan Review）；
- STPR——软件测试规程评审（Software Test Procedure Review）；
- VDR——版本描述评审（Version Description Review）；
- OMR——操作员手册评审（Operator Manual Review）；
- SMR——支持手册评审（Support Manual Review）；
- TRR——测试就绪性评审（Test Readiness Review）；
- PRR——产品发布评审（Product Release Review）；
- IPR——安装计划评审（Installation Plan Review）。

2000 年,在广泛调研研究成果和文献的基础上,Sauer 和 Jeffery[43]讨论过影响 FTR 有效性的因素。对正式技术复审的讨论将集中在:

- 参加者;
- 事先准备;
- FTR 会议;
- 推荐的 FTR 后活动。

1. FTR 的参加者

所有的 FTR 是由一位评审组长和评审组进行的,选择合适的参加者是特别重要的,因为他们有批准或不批准设计产品的权利。

1) 评审组长

因为指定合适的评审组长是影响 FTR 成功的主要因素,所以要找到这个岗位的候选人的某些特性:

- 开发这类评审项目的知识和经验(不必预先熟悉当前项目);
- 资历要高于或类似于项目负责人;
- 同项目负责人和他的小组有良好关系;
- 这个岗位是在项目组之外的。

因此,评审组长的合适候选人包括开发部门经理、首席软件工程师、另一个项目的负责人、软件质量保证单位的负责人,在某些情况下,还有顾客的首席软件工程师。

在某些情况下,项目负责人被指定为评审组长,做这种决定的主要理由是他对项目材料的管理知识,在大多数情况下,这种选择表明在专业上是不可取的。作为评审组长的项目负责人有一种倾向——不论是不是有意的——即限制评审范围和回避尖锐批评的倾向,评审组员倾向于被如此这般地选择。这种任命通常会削弱评审目标,并将面对的问题推迟到稍后的、更敏感的日期。

小的开发部门与小软件公司要找一位合适的候选人领导评审组通常会有相当的困难,针对这种境遇的一个可能解决办法是为这个职位任命一位外部顾问。

2) 评审组

整个评审组应当从项目组的资深成员、分配给其他项目和部门的资深专业成员、顾客和用户代表中选择,在某些情况,还有软件开发顾问,理想的情况是非项目成员占评审组的大多数。

一个重要而经常忽略的问题是评审组的规模。只要确保参加者中具备经验和方法的多样性,一个有效率的评审组由 3~5 人组成。过大的评审组有产生协调问题、浪费评审会议时间和降低准备水平的倾向,因为一种很自然的趋势是假设别人已经读过了设计文档。

2. FTR 的准备

FTR 会议的准备工作是由评审组长、评审组和开发组完成的,但要求每一位与会者专注于这个过程不同的方面。

1）评审组长的准备

评审组长在准备阶段的主要任务是：

- 指定组员。
- 安排评审会议。
- 向组员分发设计文档(硬拷贝、电子文件等)。

最为重要的是应当将评审会议安排在将设计文档分发给评审组员之后不久，按时开会，防止在项目组着手下一个开发阶段之前毫无道理地浪费很长时间，并因而减少偏离进度安排的风险。

2）评审组的准备

要求组员在召开评审会议前评审设计文档并列出他们的意见，在文档可裁剪的情况下，评审组长可以通过给每个组员分配一部分文档来减轻负担。

确保评审完备性的一个重要工具是检查表，除了一般的设计评审检查表之外，还有专供较常见的分析和设计文档使用的检查表，并可以在必要时构造它们。检查表通过提醒评审者所有需要注意的主要和辅助问题对设计评审做出贡献。

3）开发组的准备

开发组需要准备设计文档的一个简短讲解。假设评审组员已经透彻地读过设计文档，现已熟悉项目的轮廓，这个讲解应当集中在等待批准的主要专业问题，而不要在项目的一般描述上浪费时间。

项目负责人避免专业人员的批评和削弱评审有效性的最常见伎俩之一是全面讲解设计文档。这类讲解擅长消耗时间，它使评审组疲惫，且只留很少的讨论时间，所有有经验的评审组长知道怎样对付这种现象。

3. FTR 会议

评审组长在领导讨论和不离开议程方面的经验是成功 FTR 会议的关键，一个有代表性的 FTR 会议议程包括：

(1) 设计文档的简短讲解。

(2) 评审组员发表意见。

(3) 验证和确认讨论的每条意见，以决定项目组必须执行的措施(改正、更改和添加)。

(4) 关于设计产品(文档)的决定，它确定项目的进展。这些决定可以采取三种形式：

- 完全批准——使项目立即继续前进到下一阶段。根据情况，完全批准可伴之以要求项目组完成某些小的改正。
- 部分批准——批准项目的某些部分立即继续前进到下一阶段，而对项目的剩余部分要求采取重大措施(改正、更改和添加)。仅在措施项满意完成后才许可这些剩余部分进入下一阶段。这种许可可以由被分配评审完成的措施项的评审组员做出、由全部评审组在专门的评审会议上做出或是由评审组长在认为合适的任何其他场合做出。
- 拒绝批准——要求重复进行 FTR。这种决定适用于多重重大缺陷尤其是有关键缺陷的情况。

4. 评审后的活动

交出 FTR 报告后，需要 FTR 组或其代表跟踪改正性措施的执行情况并考察改正的部分。

1）FTR 报告

评审组长的职责之一是在评审会议后立即发出 FTR 报告。FTR 报告的及早发布使开发组能够较早进行改正性措施并使对项目进度安排产生的延迟最小，报告的主要段落包括：

（1）评审讨论简况。

（2）关于项目后续行动的决定。

（3）所需措施的完备清单——项目组必须完成的改正、修改和添加。对于每个措施项，列出预计完成日期和负责的项目组成员。

（4）分配跟踪改正性措施进行的评审组员的名字。

2）跟踪过程

被指定跟踪改正性措施的人——在许多情况下，是项目负责人——确定将每个措施是否圆满完成作为项目继续到下一阶段的条件。跟踪情况应当被完整地记录，以便在将来需要时能够澄清这些改正。

令人遗憾的是，FTR 报告的全部或部分常常是无价值的，或是因为不充分准备的评审组，或是因为有意避开彻底的评审。从 DR 报告的下列特性相当容易识别出这种情况：

- 极短的报告，只限于设计产品的公文式批准，未列出检测出的缺陷。
- 短报告，批准完全继续到下一项目阶段，列出了若干小缺陷，但无措施项。
- 列出了若干不同严重性的措施项的报告，但没有指示跟踪（改正安排等），没有可用的书面跟踪行动。

Pressman 列出过完成一份成功 FTR 的指南，重点在基础设施、FTR 的准备和会议的召开，Pressman 关于正式技术复审的黄金“指南”也适用于审查和走查会议。

（1）设计评审的基础设施：

- 为每种设计文档建立检查表，或至少为常用文档建立。
- 训练资深专业人员处理重大技术问题以及评审过程问题，把受过训练的专业人员用做 FTR 组的储藏资源。
- 定期分析过去. FTR 对缺陷检测的有效性，以改进 DR 方法。
- 将 FTR 作为项目活动计划的一部分安排，并分配所需的资源，这要作为软件开发机构的标准操作规程的不可缺少的组成部分。

（2）设计评审组：评审组应当有规模限制，通常最好 3～5 名成员。

（3）设计评审会议：

- 以建设性的方式讨论专业问题，同时防止把这些问题个人化，这要求保持气氛没有不必要的紧张。
- 遵守会议议程。脱离计划好的议程总是会干扰评审的效率。
- 专注于通过验证和确认与会者的意见来检测缺陷，避免讨论针对检测出缺陷的可

能解决办法，以节省时间，免得离开议题漫无边际地闲扯。

- 在对于错误的重要性有不同意见的情况下，最好是通过记下问题并将它移交另外的会议去讨论来结束争辩。
- 正确地将讨论记入文档，特别是参加者的意见细节和他们的验证与确认结果。如果这个文档被用做编制评审报告的输入或基础，这个步骤就特别重要。
- 评审会议期不应超过2小时。

(4) 评审后活动：

- 编制总结讨论的问题和措施项的评审报告。
- 建立跟踪过程，以确保措施项清单中的所有改正得到满意实施。

正式技术复审过程如图10-7所示。

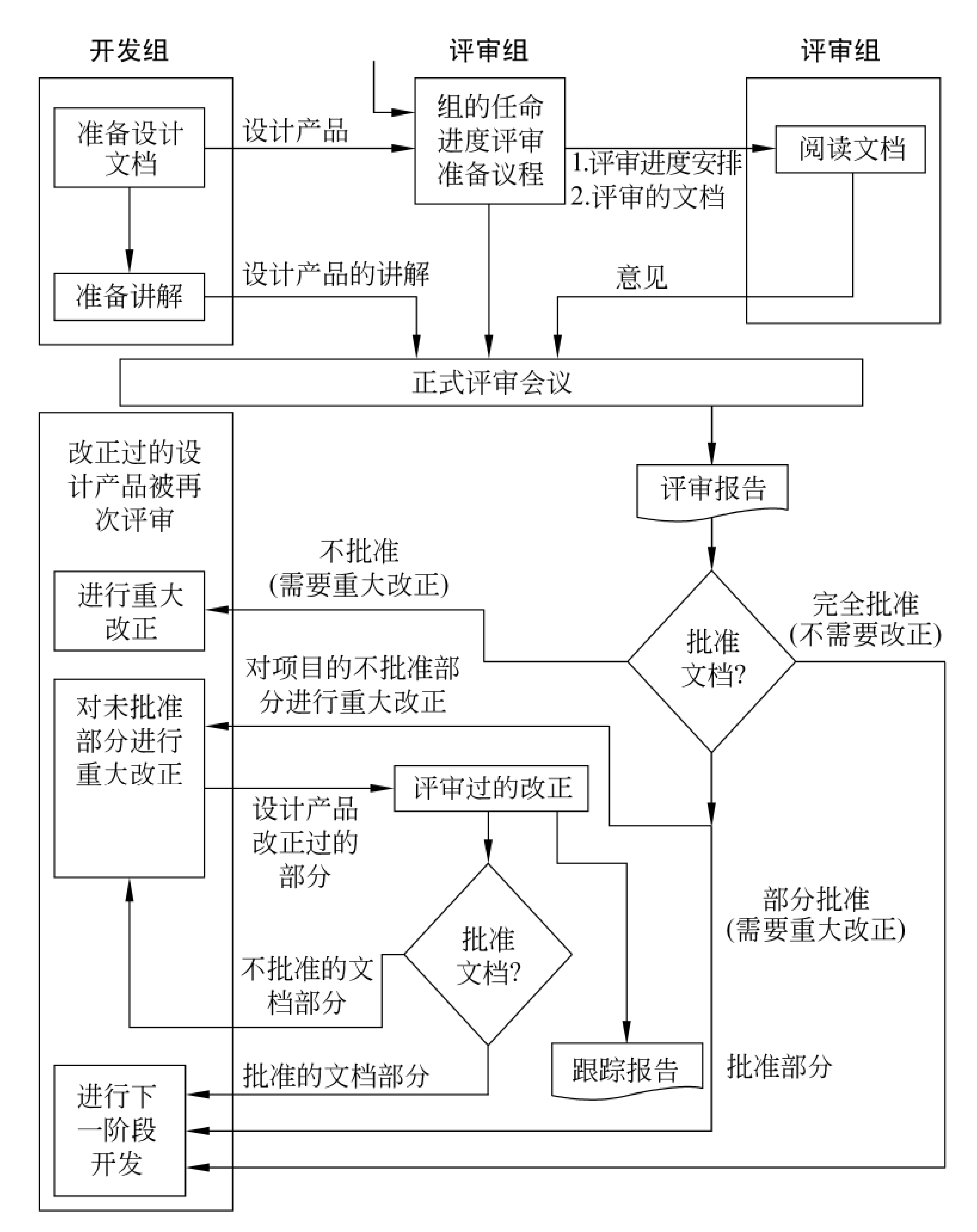


图 10-7 正式技术复审过程

10.3.3 同行评审

本节讨论两种同行评审方法——审查(inspection)和走查(walkthrough)。正式技术

复审和同行评审之间的主要不同在于他们的参加者和权限。FTR 的大多数参与者的职位比项目负责人和顾客代表要高,同行评审的参与者是项目负责人的同级、部门或其他单位的成员。另一个重大的不同在于权限和每个评审方法的目标。正式技术复审的权力是批准设计文档,以便项目下一阶段的工作能开始;这个权力没有给予同行评审,同行评审的主要目标在于检测错误和对于标准的偏离。

今天,一方面是计算机化设计工具(包括 CASE 工具)的出现;另一方面是巨大软件系统的出现,一些专业人士倾向于减低诸如审查和走查之类的人工评审的价值。然而,过去的软件调查乃至最近的实验性研究发现:同行评审是有实效的方法,也是高效的方法。

走查和审查的区别是其正式性的等级,审查是二者之中更为正式的,审查强调改正性措施的目标。走查的发现限于被评审文档的意见,而审查的发现还同改进开发方法自身的工作相结合。所以,审查与走查相比,审查被认为对一般级别的 SQA 做出更重大的贡献。

通常,审查基于全面的基础设施,包括:

- 建立为每种设计文档以及编码语言和工具开发的审查表,它们被定期更新。
- 基于以往的发现,建立典型缺陷类型的频度表,以指引审查者注意潜在的"缺陷聚集区"。
- 在审查过程问题方面培训干练的专业人员,这个过程使他们能够作为审查组长(协调员)或审查组成员。受过培训的雇员们作为未来项目可用的专业审查员的储藏资源。
- 定期分析以往审查的有效性,以改进审查方法学。
- 将计划安排的审查引进项目活动计划里面,并分配所需资源,包括改正检测出的缺陷所需的资源。

这里描述的审查和走查过程是这些方法的较常用的版本。开发机构常常修改这些方法以适应地方性。应当注意,由于这些可变性,尤其是在走查过程中,两个方法之间的不同容易模糊。因此,有时候一些专家将走查看作审查的一种,反之亦然。

关于哪种方法更好的争论还没有解决,基于对每种方法的调查研究,Gilb 和 Graham[44]下了一个结论:作为审查的一种替代方法,走查显示出"以同样的费用发现的缺陷却少得多"。

关于同行评审方法的讨论集中在:

- 同行评审的参加者;
- 同行评审的必要准备;
- 同行评审会议;
- 同行评审后的活动;
- 同行评审的效率。

设计和代码审查作为过程性模型,最初是由 Fagan 于 1986 年描述并正式化的。至于走查,Yourdon 于 1979 年就提出了有关原理和过程的透彻而详细的讨论[45]。

1. 同行评审的参加者

优化的同行评审组有 3～5 个参加者，在某些情况下，增加 1～3 个参加者也行。所有参加者应当是软件系统设计者的同事，对同行评审的成功做出贡献的一个重大因素是小组的“交融”（审查和走查之间有所不同）。

推荐的同行评审组包括：

- 一位评审组长；
- 作者；
- 特定专业的人员。

1）评审组长

评审组长的角色（在审查中是“仲裁员”，在走查中是“协调员”）按同行评审的类型稍有不同。这个岗位的候选人必须做到以下几点：

（1）精通当前评审类型的项目的开发并熟悉其技术，不必对当前项目预先熟悉。

（2）同作者和开发组保持良好关系。

（3）来自项目组外。

（4）显示出在专业会议的协调和领导方面有被证实的经验。

（5）对于审查，还需要受过仲裁员培训。

2）作者

作者总是各种同行评审的一位参加者。

3）特定专业的人员

特定专业的人员参与两种同行评审方法的问题因评审的不同而不同。

对于审查工作而言，应推荐以下专业人员：

- 一位设计人员：负责被评审软件系统的分析和设计的系统分析员。
- 一位编码人员或实现人员：他应是熟悉编码任务的专业人员，最好是任命的编码组长。这位审查员必须能将他的专长用于检测那些可能导致编码错误和软件实现困难的缺陷。
- 一位测试人员：他应是一位有经验的专业人员。最好是任命的测试组长，他专注于识别在测试阶段常常检测出的设计错误。

对于走查工作而言，应推荐以下专业人员：

- 一位标准推行员：这位组员的专长是开发标准和规程，被分配的任务是指出那些偏离标准和规程的地方。这类错误对小组的长期有效性有重大影响，这是因为这类错误对新成员加入项目组造成额外困难，而且它们将降低系统维护组的有效性。
- 一位维护专家：他的关注重点是可维护性、灵活性和可测试性，并检测那些能妨碍缺陷的改正或进行未来更改的设计缺陷。他还具有文档编制方面的专长，文档的完备性和正确性对任何维护活动都是至关重要的。
- 一位用户代表：在走查组中，内部用户（当顾客是同一公司里面的一个单位时）或外部用户代表对评审的有效性做出贡献，因为他是以用户和顾客的观点而不是从

设计者——供货商的观点来考察软件系统的。在没有“真正的”用户可用时(例如在开发COTS软件包的情况),组员可以担当这个角色,并通过比较原始需求同实际的设计来关注有效性问题。

4) 组的分工

召开评审会议自然需要给组员分派具体任务,有两个组员担任文档的讲解员和书记员,后者将讨论记入文档。

(1) 讲解员:在审查会议期间,文档讲解员是由仲裁员指定的;通常,讲解员不是文档的作者,在许多情况下,软件编码人员充当讲解员,因为他可能是对设计逻辑及编码含义理解最好的组员。相反,对于大多数走查会议,作者正是最熟悉文档的专业人员,他被选来在小组里讲解。有些专家认为将作者指派为讲解员可能影响小组成员的公正性。

(2) 书记员:评审组长常常——但不是总是——担当会议的书记员,并记录那些已经指出的缺陷,待开发组改正。这个任务不是事务性的;它要求对讨论的问题有透彻的专业理解。

2. 同行评审会议的准备

评审组长和评审组员应该勤奋地完成他们的准备,评审类型决定准备的范围。

1) 同行评审组长对评审会议的准备

评审组长在准备阶段的主要任务是:

(1) 同设计文档作者一起确定要评审文档的哪些章节。这些章节可以是:

- 最困难和最复杂的章节;
- 最关键的章节,其中的任何缺陷都可能对程序应用、甚至用户造成严重的损害;
- 易出错的章节。

(2) 选择组员。安排同行评审会议。限制评审会议每天不超过两次,每次不超过2小时是可取的。当评审任务相当大时,应当安排若干次评审会议。重要的是在有关设计文档已准备好提交审查之后不久就安排会议。这种贴近有助于使(基于在安排的评审中可能发现有缺陷的文档部分的)设计追加物的范围和/或数量最小。此外,为使过程平滑,评审组长应当为他的组安排一次纵览性会议。

(3) 在评审会议前将文档发放给组员。

2) 同行评审组为评审会议做的准备

审查组员需要做十分全面的准备,而对走查组员来说,所需的准备较简单。

要求审查组员阅读待评审的文档章节,并在审查会议开始前列出他们的意见,这种事先准备的目的是保证会议的有效性。也可能要求他们参与一次纵览性会议,在这次会议上,作者向审查组提供评审所选文档章节必要的有关背景:项目概况、逻辑、处理、输入、输出和接口。如果参加者已经很熟悉材料,可以免去这个纵览性会议。

支持审查员评审的一个重要工具是检查表。在成熟的开发部门里,人们可以发现用于较常见类型的开发文档的专门检查表。

在走查会议前,组员简要地阅读材料,以获得对待评审章节、项目及其环境的一般性

概况。缺乏关于项目及其重要方面预备知识的参加者会需要多得多的准备时间。在大多数采用走查的机构里，审查组参加者不需要事先准备他们的意见。

3. 同行评审会议

典型的同行评审会议采取下面的形式：讲解人读文档的一个章节，如果需要，再用他自己的话简要说明所涉及的问题。随着会议的进行，参加者或者交出他们对文档的意见，或者说出他们对意见的反应。这种讨论应当限于识别错误，这意味着不应当讨论没有把握的解决办法。与审查会议不同，典型的走查会议的议程以作者的简短讲解或项目和待评审设计章节的概述开始。

会议期间，书记员应当将识别出的每个错误的位置与描述、类型和特征（不正确、遗漏工件、多余工件）记入文档。审查会议书记员将补充每个缺陷的估计严重程度，这是一个对已发现缺陷进行统计分析和形成预防性和改正性措施有用的因素。表 10-2 提供了给错误严重程度分类的一个可接受的框架。

表 10-2　设计错误按严重程度的分类

严重程度	描　　述
5（关键的）	（1）妨碍建立根本能力 （2）危及安全、保密或其他关键需求
4	（1）严重影响根本能力的建立，不知道回避的办法 （2）严重影响项目或系统维护的技术、费用或计划安排风险，不知道回避的办法
3	（1）有害地影响根本能力的建立，知道回避的办法 （2）有害地影响项目或系统维护的技术、费用或计划安排风险，知道回避的办法
2	（1）用户（操作者）不方便使用，但不影响所需的使命或操作上的根本能力 （2）开发或维护人员不方便使用，但不妨碍他们履行职责
1（轻微的）	其他缺陷

至于审查和走查会议的长度，同 FTR 的规则相同：会议不应超过 2 小时，也不应在一天里安排两次以上的会议。

4. 会议文档编制

在审查会议末尾产生的文档要比走查会议的文档全面得多。在审查会议之后产生两个文档并随后分发给会议参与者：

（1）审查会议发现报告（inspection session findings report）。由书记员产生的这份报告应当是完全的，并在会议闭会后立即分发，其主要目的是确保已识别出的待改正和跟踪的错误全部记入文档。

（2）审查会议总结报告（inspection session summary report）。这份报告由审查组长在讨论同一文档的会议或一系列会议之后很快编写出来。这种类型的典型报告总结审查发现和审查中投入的资源；它同样提出基本质量和效率度量。这份报告主要用做分析工作的输入信息，这种分析旨在审查过程改进和特定文档或项目之外的改正性措施。在一次会议或一系列走查会议结束时，应将错误记录文档的副本——“走查会议发现报告”

交给开发组和会议参加者。

5. 同行评审后的活动

区分这里讨论的两种同行评审方法的一个要素是同行评审后的问题。与走查相比，审查过程并不以一次评审会议或分发报告而结束。要进行审查后的活动以证实以下情况：

(1) 设计小组对评审组长(或其他的组员)在分配的跟踪活动中所发现的所有错误进行了迅速、有效的修改。

(2) 将审查报告转送给内部的改正措施委员会(Corrective Action Board,CAB)以供分析。这样可以启动改正性和预防性措施以减少未来的缺陷数和提高生产效率。

在图 10-8 和图 10-9 中显示了同行评审方法、参加者和过程要素的比较。

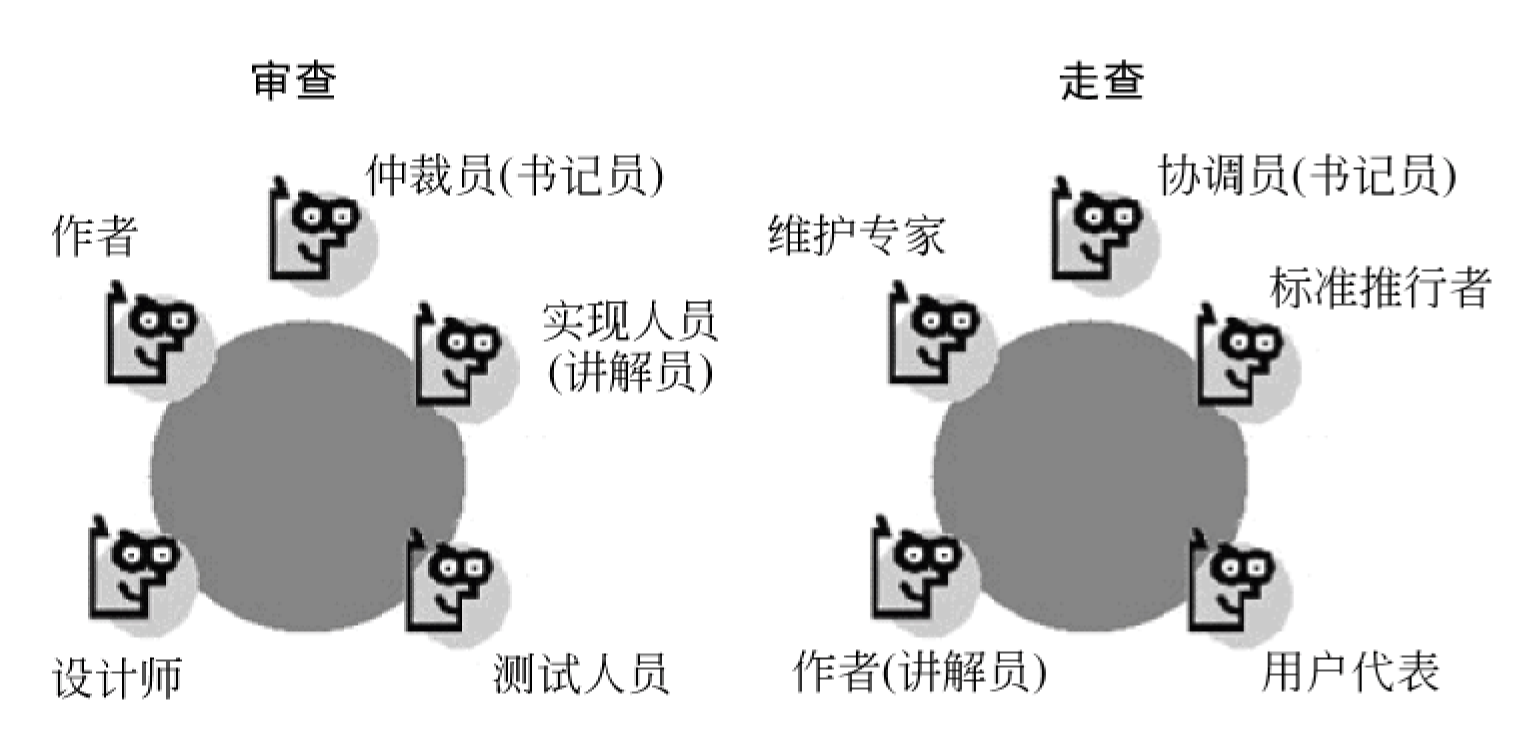


图 10-8　审查与走查参加者的比较

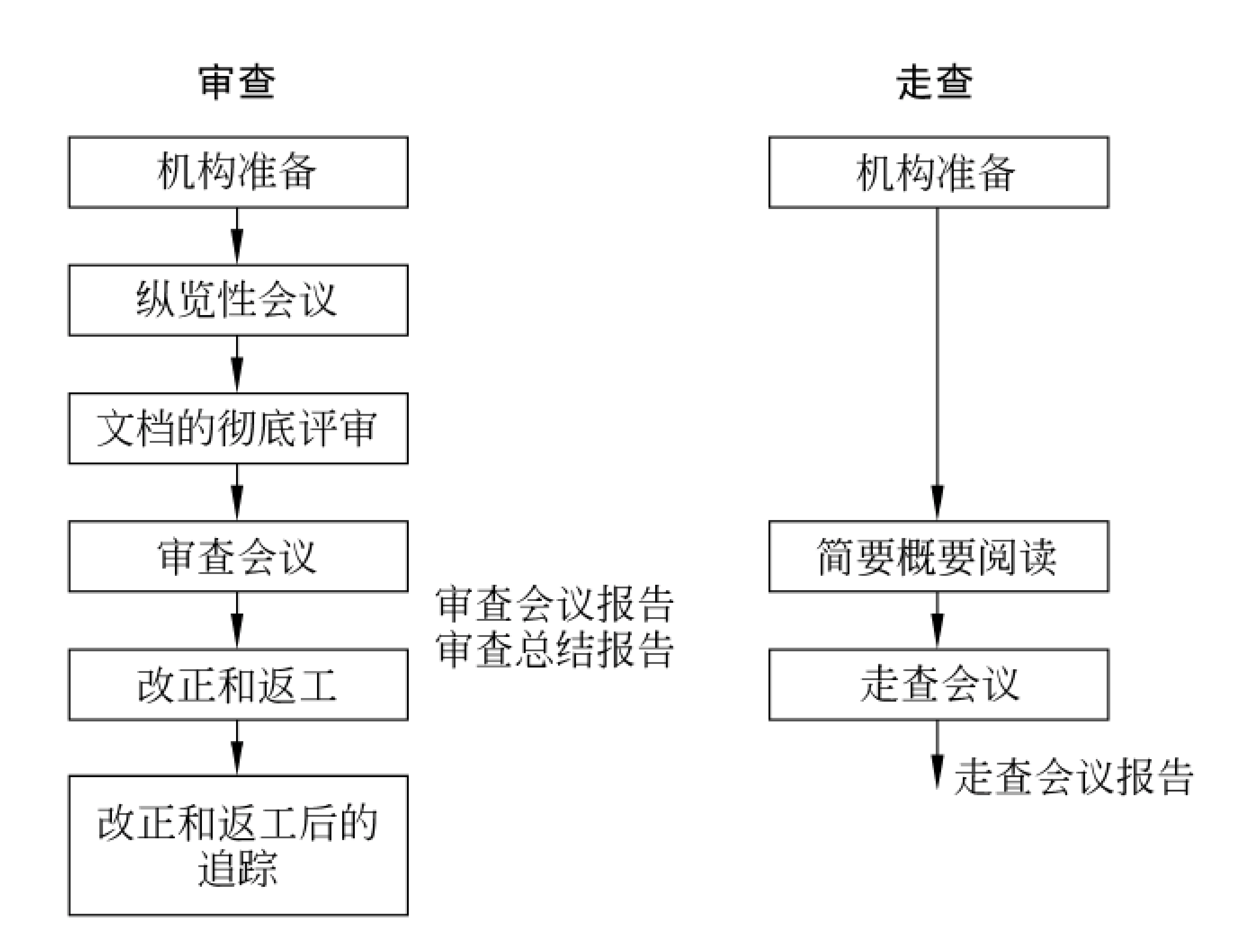


图 10-9　审查与走查过程的比较

6. 同行评审覆盖

文档和总代码量中虽然只有部分进行同行评审，覆盖文档页的 5%～15%，仍然代表了对整个设计质量的贡献。因为确定同行评审对总体质量贡献的不是覆盖页数的百分比，而是选择了哪些页。重要的是，随着重用软件使用量的增长，要求审查的文档页数和

代码行数明显下降。下面列出推荐和不推荐在同行评审中评审的文档段落：

1）推荐进入同行评审的段落

- 复杂逻辑的段落；
- 关键段落，这里的缺陷严重损害基本的系统功能；
- 同新环境有关的段落；
- 由新的或无经验组员设计的段落。

2）不推荐进入同行评审的段落

- “直接明了的”段落（不复杂）；
- 在开发组过去的类似项目中评审过若干次的那种段落；
- 预期如果有错误也不影响功能的段落；
- 重用的设计和代码；
- 设计和代码的重复部分。

10.3.4 评审方法的比较

对实践人员和分析员来说，本节讨论的评审方法的比较应当有助于他们的工作（见表 10-3）。

表 10-3 评审方法学的比较

性　质	正式技术复审	审　查	走　查
主要直接目标	（1）检测错误 （2）识别新风险 （3）批准设计文档	（1）检测错误 （2）识别对标准的偏离	检测错误
主要间接目标	知识交换	（1）知识交换 （2）支持改正性措施	知识交换
评审组长	首席软件工程师或资深员工	培训过的仲裁员（同事）	协调员（同事，偶尔是项目组长）
参加者	顶层员工和顾客代表	同事	同事
项目组长参加	是	是	是（通常作为评审倡导者）
组中特定专业的人员		（1）设计者 （2）编码人员或实现人员 （3）测试人员	（1）标准推行者 （2）维护专家 （3）用户代表
评审过程：			
• 概览性会议	无	有	有
• 参加者的准备	有——透彻	有——透彻	有——简要
• 评审会议	有	有	有
• 改正的跟踪	有	有	无
基础设施：			
• 参加者的正式培训	无	有	无

续表

性质	正式技术复审	审查	走查
• 检查表的使用	无	有	无
有关错误的数据收集	无正式需要	有正式需要	无正式需要
评审文档	正式技术复审报告	(1) 审查会议发现报告 (2) 审查会议总结报告	走查会议发现报告

10.4 软件测试管理

在前面介绍了软件测试是软件质量保证的关键步骤。为了真正做好软件测试工作,系统地建立一个软件测试管理体系是非常重要的,只有这样才能确保软件测试在软件质量保证中发挥应有的关键作用。

建立软件测试管理体系有以下几个方面:

(1) 确定软件测试的每个阶段:制定测试计划,测试设计,实施测试,建立和更新测试文档以及测试管理。

(2) 确定阶段间的相互关系。制定的测试计划、测试设计、实施测试是按顺序依次进行并且相互作用,阶段间衔接是规范化的,每个阶段有开始和结束标志。测试管理是对这三个阶段进行监督和管理。建立和更新测试文档贯穿整个测试流程。

(3) 确定进行各阶段测试所需要的标准和策略,掌握其相关文档。

(4) 确定监督、管理和控制各测试阶段的准则和方法。

(5) 确保可以获得必要的资源和信息,以支持测试流程的正常进行和监督工作的顺利开展。

(6) 为了提高测试质量,适当改进措施。

软件测试管理的主要内容如下:

(1) 软件产品的监督和测量

对软件产品的质量特性进行监督和测量,主要依据软件需求规格说明书,验证产品是否满足要求,所开发的软件产品是否可以交付,要预先设定质量度量指标并进行测试,只有符合预先设定的指标才可以交付。

(2) 对不符合要求产品的识别和控制

对于软件测试中发现的软件缺陷,要认真记录它们的属性和处理办法,并进行跟踪,直至最终解决,在修复软件缺陷之后,要再次进行验证测试。

(3) 软件过程的监督和测量

从软件测试中可以获取大量关于软件过程及其结果的数据和信息,它们可用于判断这些过程的有效性,为软件过程的正常运行和持续改进提供决策依据。

(4) 产品设计和开发的验证

通过设计测试用例对需求分析、软件设计、程序代码进行验证,确保程序代码与软件设计说明书一致,软件设计说明书与需求规格说明书一致。对于验证中发现的不合格现

象，同样要认真记录和处理，并跟踪解决。解决之后，也要再次进行验证。

10.4.1 测试团队和开发团队的协作

图 10-10 表示的是软件开发和测试过程中两种典型的组织结构。在强调开发的软件组织中，开发经理负责开发人员和测试人员的管理，而在一般的软件企业中，测试组常常独立于开发组，单独设置，以软件开发组长为首的开发部门和以软件测试组长为首的测试部门既各有分工又需要相互合作，确保软件质量符合设计标准。在大型的软件企业中，开发和测试部门可能会设置各自的经理，和项目经理三足鼎立，共同开发软件。

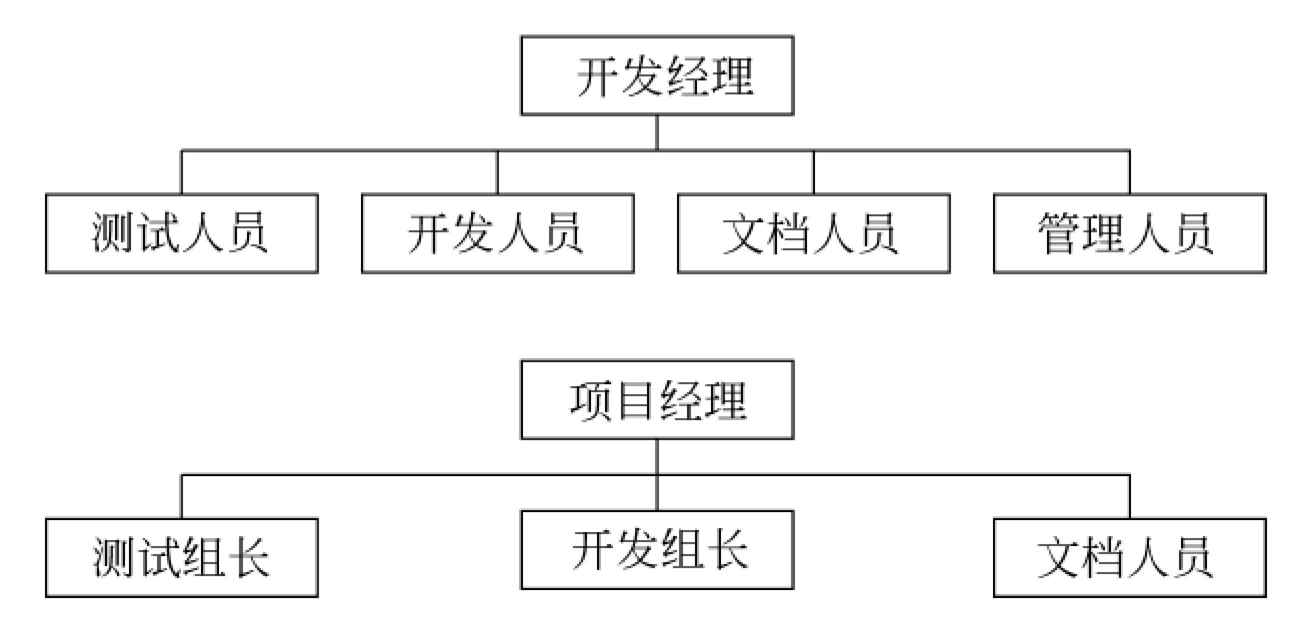


图 10-10　两种软件企业的组织模型

测试工程师和开发工程师承担的是开发工作的两个不同方面，一个是创建，另一个是破坏。虽然两者的最终目的都是一样的，但在达成目标的方式上却有很大的差异。因此，在为同一个目标奋斗的过程中，发生冲突也是难免的，但通过下面的一些建议，换个视角看看开发人员的生活和工作，可能很多的冲突就能化解于无形。Cem Kaner 在 *Testing Computer Software* 一书中说过："最好的测试人员不是发现最多 bug 或是使得最多开发人员不自在的人，而是能够说服开发人员修正 bug 最多的人。"

资深的测试工程师关河[46]把和开发人员交流的经验归结为"五要四不要"。

1. 五要

1）要耐心和细心

细心是测试工程师的一个基本素质，测试工程师是对质量负责的人，涉及到质量问题，就不能含糊，因此一定要细心，细心对待每一个可能的 bug、细心对待每一段被你检查的代码，细心对待每一个你撰写的 bug 报告，细心对待你发出的每一封邮件。细心是一种态度，你的态度迟早会感染和你合作的开发人员，而这往往是合作愉快的基础。

在测试工程师的工作经历中，不厌其烦地向开发人员解释一个 bug，让他认识到 bug 的重要性是经常的事情。对任何人来说，被人指出自己的缺点和不足都不是让人舒服的事情，因此，一点不耐烦的情绪就可能引起对方很大的反感，给自己的工作带来不必要的麻烦。

2）要懂得尊重对方

开发是一件需要全面和综合考虑的工作。由于各种原因导致程序中出现问题是很正常的现象，作为测试工程师，发现了这些问题并不值得你夸耀，也不能说明你比开发工

程师聪明。一个好的测试工程师一定是懂得尊重开发工程师的人,尊重对方的技术水平,尊重对方的代码。一般来说,对他们最大的尊重就是承认他的专业水平,承认他的代码。因此,记得在合适的时候表达对他的尊重。

3）要能设身处地地为对方着想

开发工程师一般都处在较大的工作压力下,项目经理直接考核他们的指标很大程度上是已完成的代码,所以在工作任务紧张的时候,对于测试工程师报上来的 bug 会拖延解决甚至推脱,给测试工程师的感觉就是很不合作。那么在这个时候,就需要设身处地地为对方着想了,每个人都会为自己的工作在内心排定优先级,如果他认为解决你发现的 bug 不是重要的事情,那么最大的可能就是你并没有向他解释清楚这个 bug 的严重程度。

发现 bug 是测试工程师的责任,敦促 bug 得到解决是测试工程师更重要的责任,因此,测试工程师可以心平气和地和开发人员坐下来讨论一下 bug 的严重程度,和他一起排定 bug 的优先级别并确定解决的时间。

4）要有原则

不要忘记,测试工程师需要对产品的质量负责,在这一点上一定要有原则。测试工程师可以和开发工程师建立良好的个人关系,但在具体的事情上,一定要按照公司的相关流程来处理。当然,在坚持原则的同时,可以采用一些委婉的表达方式,可以在允许的情况下尽量体谅开发工程师。但请记住,一个有原则的测试工程师才能真正帮助开发工程师,才能赢得开发工程师的尊重。

5）要主动承担

如果开发工程师要求你承担部分不属于你的责任,例如,定位你发现的 bug 到代码一级,或者是帮助他编写部分文档和代码。我们的建议是在可能的情况下尽量多承担。其实都是工作上的事情,有能力的话,多做一点也无妨。但在主动承担时,一定要明确是在自己确有余力的情况下才能去承担;否则,婉拒是最好的对策。

2. 四不要

1）不要嘲笑

不要嘲笑你所发现的 bug,即使是非常愚蠢的错误也绝对不要嘲笑,说不定那个错误是因为开发工程师连续加班 24 小时后犯下的。对别人的工作始终应该尊重。如果你觉得有必要提醒他不再犯一些经常犯的错误,可以采用这样的方式:编写一份测试过程中发现的开发人员常犯错误的文档(记住,千万不要写上谁犯了这些错误),用轻松的口气调侃一下,发送给开发人员。

2）不要在背后评论开发工程师

永远不要在背后评论开发工程师的技术能力,这绝对是个非常忌讳的事情。一时的口舌之快或许会使你永远不再能同他良好地合作。要知道,开发工程师最在意地就是别人对他的技术能力的评价。其实这不仅仅是作为测试工程师的准则,也应该是做人的准则。

3）不要动辄用上层来压制对方

在出现和对方的意见分歧的时候,应该采用什么方式说服对方呢？直接向上层求助

当然是一个办法，但这种办法带来的负面作用也是很明显的，首先是作为上层的处理结果可能不一定符合你的愿望（在很多公司，开发工程师的地位高于测试工程师的地位，这种地位的不平等导致上层在处理分歧时会有一定的偏向性）；其次是动辄拿出上层来压制对方只能给他人留下无用的印象。所以在出现分歧时，尽量尝试通过沟通来解决，实在不行，再动用最后的手段。

4）和开发人员的沟通不要只有 bug

除了在 bug 记录单上，在其他的地方也要让和你合作的开发工程师接触到你，午餐或是集体活动的时候多和对方聊聊天，一方面可以增进彼此的感情，混个脸熟，打交道的时候也方便；另一方面，从开发工程师那里了解业务的知识和他负责模块的方方面面，对自己也是提升。

其实关键的就是两点：首先是多从别人的角度去想想，所谓“换位思考”，多尊重对方就一定能得到对方的尊重与配合；其次是加强和开发工程师的沟通，让他清楚地认识到你的工作对他的价值，你发现的每一个 bug 的重要性。一个好的测试工程师一定是在公司里被所有人尊重的快乐分子，而不应该是一个“铁面判官”。

10.4.2 测试人员应具备的素质

在软件行业中有种普遍的存在的现象，那就是让那些无经验新手、没有效率的开发者或不适合干测试工作的人去做测试工作。这绝对是一种盲目行为，一方面这种现象造成了行业不重视软件测试的恶性环境；另一方面造成了中国软件行业输出质量普遍低下，使我们无法参与到国际软件生产中。对一个软件系统进行有效法的测试所需要的技能绝对不比进行软件开发需要的少，事实上测试人员在工作中将遇到许多开发者不可能遇到的问题。

Bill Hetzel 在 *The Complete Guide to Software Testing* 一书第 11 章节中罗列了 5 项优秀测试人员所拥有的重要特征：Controlled（可管理，有条理）、Competent（掌握测试技术的）、Critical（专注于发现问题的）、Comprehensive（注重细节的）、Considerate（能够和开发人员很好交流的），简称为测试人员的 5C 职业特征。参考专家们所写的，这里总结和扩展了测试人员应该具备的 10 项职业素质：

1. 沟通能力

测试人员必须能够同测试涉及的所有人进行沟通，具有与技术人员和非技术人员交流的能力。用户和开发这两类人几乎没有什么共同语言，所以需要测试人员作为一种良好的中间接触媒介。和用户谈话的重点必须放在系统可正确地处理什么和不可处理什么上，而和开发者谈相同的信息时，就必须将这些活动重新组织以另一种方式表达出来。

2. 共同价值观

用户担心将来使用一个不符合自己要求的系统，开发者则担心由于系统要求不正确而使他不得不重新开发整个系统，公司则担心这个系统无法得到用户的认可而使它的利益受损。测试人员必须和每一类人打交道与他们形成共同的价值观，具备了这种能力可

以将测试人员与相关人员之间的冲突和对抗减少到最低程度。

3. 技术能力

开发人员轻视那些不懂技术的人是一种普遍存在现象。测试人员必须很好地理解被测试软件概念,会使用其中的一些重要工具,做到这一点需要有几年以上的内行编程经验,每一位测试人员都应该具有一个技术背景,还应该具有一定的职业成熟经验。这些有利条件可以帮助测试人员对软件开发过程有更深入的理解,从开发人员的角度发现被测试软件中存在的问题,简化自动测试工具的学习曲线。

4. 自信

测试人员和开发人员的工作性质相反,所以经常出现开发人员指责测试人员工作出现错误的事情。测试人员必须对自己的观点保持足够的自信,如果容许别人对自己的工作结果指东道西,就不能完成更多的事情了。

5. 交流

当你告诉某开发人员的程序出现了错误时,需要用一种婉转且留有余地的商讨口吻和开发人员交流。如果采取的方法过于强硬,对测试人员来说后果是可怕的,在以后和开发部门的合作方面就相当于“赢了战争却输了人心”。例如,你可以经常和开发人员在公司的咖啡室对项目中发现的问题进行交流,在放松的环境里讨论问题有事半功倍的效果。

6. 记忆

测试人员应该具备将历史类似错误从记忆深处挖掘出来的本领,这一能力在测试过程中的价值是无法衡量的。出错类型无非就这么几种,我们需要处理它们的大量变种。

7. 耐心

测试工作需要耐心,有时我们需要花费大量的时间去剥离、确认和跟踪一个错误,测试工作是那些坐不住板凳的人无法胜任的。

8. 怀疑

怀疑也是测试人员必须具备的基础品德,开发人员本能地掩盖所有已经出现的或者未出现的错误。测试人员在听取了开发人员的说明后,必须保持怀疑态度直到经过自己认真的核实。

9. 自我激励

没有比干测试更容易让人意志消沉的工作了,看了这么多的错误很容易产生自己的存在是不是也是一种错误的想法,只有那些具有自我激励技能的人员才能够使自己每天的工作持续下去。

10. 洞察力

准确地捕获用户观点，极限地追求软件输出质量，对某些细节的敏感程度综合起来就体现了测试人员的洞察能力。洞察力让测试人员透过问题的表象深入到问题的内部，越接近问题的本质就越容易解决问题的存在。

10.4.3 如何成为一名优秀的测试工程师

要成为一名优秀的测试工程师除具备上述素质外，还得有两项特征：

1. 要有梦想

成为一个优秀的测试工程师是一个渐进的过程，这也许是所有软件测试人员的一个梦想。梦想会激励你的灵魂，左右你的行为，并给你一种不断向前的力量。

有很多优秀的测试人员，他们擅长发现问题，有丰富的工作经验，能熟练地组合各种测试方法。但是，他们缺乏勇气去承认，承认自己是名优秀的测试人员，同样的问题也在一些技术交流会议中出现。技术交流会中有很多有意思的人，他们有自己的观点和良好的从业经验。但是，当鼓励他们在会议上发言或者写相关的技术文章在论坛上发表的时间，他们中的大多数会说："我不是一个专家，不能肯定这个回答是否正确。"现在技术论坛上发表的文章品质低劣，很大的原因在于我们的测试人员过于"害羞"。当很多只有一点测试经验的人发言，写文章表达他们的思想的时候，大量的优秀观点却因为这些优秀测试人员的自我怀疑而深埋在他们自己的心中。

一旦具备了梦想和勇气这还不够，还需要形成自我思想。一个没有自我思想的测试人员永远都成为不了专家。很多测试行业的从业人员只会积极跟别人索取，拿来就用根本不去想想别人的用意。他们不善于思考拿来就用，不善于改进只会生搬硬套，这真是测试行业的不幸。

总之，专家梦想的实现是多样的，你可以借鉴别人的经验，有所选择地复制部分，但是你绝对不能没有自己的实现思想。有了梦想，具备了勇气和思想才是一个好的开端，下面的问题就是它的实现。例如，假如我奇怪的个性造成了别人的反感，我就不能很好地融入团队；假如我对测试技术的认识没有足够的深度，我的测试设计就会很脆弱；假如我只求当前的工作成果不思进取，我就会作很平庸地工作等，因此，成为测试专家之路真是险阻重重。

真正优秀的测试工程师有个习惯，看到任何产品都会询问自己："我能测试它吗？我能在没有规格标准的情况下测试它吗？假如只给我正常测试所需时间的一半，我应该测试些什么？""我能测试它吗？"代表专家的职业特点，"我能在没有规格标准的情况下测试它吗？"代表专家的技术能力和行业知识高超，"假如只给我正常测试所需时间的一半，我应该测试些什么？"代表专家的优秀风险回避意识。笔者跟多个测试专家聊过，总结了以下这些专家所共有的特征：

- 敢于声明他是所有专家中做得最好的。姑且我们不论好坏，至少我们已经知道专家的内心充满了勇气。

- 善于使用简单的、便于使用的格式递交有用的结果。
- 熟练的组合各种测试方法以适合不同项目业务要求。
- 有回旋余地地利用测试工具和资源，测试专家并不完全依赖于测试工具。
- 为达到和项目团队的完全融合，而牺牲自我个性。
- 能言善辩，在项目中懂得如何保护自己和团队。
- 把工作中存在的风险性和有限性作为建议提供给客户。
- 必要的时候劝告客户，怎样做可以做得更好。
- 诚实并且充满了正义感地为客户服务。

成为测试专家是一条漫长的修行之路，不过当你具备了上面所有的特征时，你就是专家，并且是非常优秀的测试专家。

2. 一个优秀的测试人员必然是个优秀的开发人员

这也是测试论坛上最有争议的话题，一种强烈支持软件测试人员需要有编程经验，另一种认为软件测试人员无编码经验也可以做得很好。本书认为测试人员具备编程经验应有所选择，因为软件项目各个阶段对测试人员有所不同的工作要求。比如当前我们采用的 RUP 软件开发过程中，测试主要有 5 个阶段：

1）需求阶段

需求阶段的测试工作表现为需求复审会议，要求测试人员具备项目所涉及的行业知识、良好的客户沟通、熟练的工具使用技能等综合素质，因为本阶段加入了对 UML 用例图和顺序图的运用，所以测试人员需要对大量的 UML 图例进行业务复审。

2）设计阶段

设计阶段的测试工作表现为设计复审会议，复审会议综合了对设计模型、数据模型、界面原型、事件驱动模型等工作的复审工作，该阶段对测试人员的综合素质要求最高。

3）实施阶段

让一个没有半点编码经验的人做单元级测试是个天方夜谭，没有编程经验的测试人员根本无法把握这个阶段的工作。虽然大部分人认为实现人员应该全面负责单元测试工作，但是作为测试人员，你就真的放心他们所做的吗？你还是需要经常进行代码抽查之类的测试活动。

4）测试阶段

这个阶段的某些行为是新入测试行业人员或者不成熟项目管理者的一个精神寄托，比如所谓的简单粗糙的功能测试。本阶段对测试人员有更高级的要求，比如开发一些测试工具或者测试驱动程序，如果没有编程经验怎能胜任此项工作？同样是两个测试人员，A 有编程经验而 B 没有，我们可以这么说：A 比 B 有更高的代码认知，A 可以用他的编程经验提前确认系统中可能存在的隐患。

当前软件开发融入了面向对象的思想，这对测试人员提出了更高的要求。测试人员除了要掌握更多的测试方法，还需要有正确的面向对象的思想，否则测试工作是出力而无功。

5）部署阶段

部署阶段面临项目或者产品的最终交付，测试工作的重要性尤为突出。如果输出是

组件形式，我们需要进行装配测试，这要求测试人员有编写模拟测试环境的能力。如果输出是产品形式，我们要做不同操作系统的安装测试和产品适应性测试，这要求测试人员有多系统的操作能力。如果输出是同产品不同版本形式，我们要分离不同版本产品做版本差异性测试。

当前，国内软件质量低下的原因，首先是我们的软件过程混乱，其次就是没有真正职业的测试人员。而国外的大部分测试人员都是从开发转过来的，这些人员从业经验丰富，能预知什么错误最容易出现，最可能出现的位置以及如何预防。其次这也关联到了测试用例的设计，好的测试用例设计是成功测试的基础。

第11章 chapter 11

软件测试案例

11.1 案例概述

对于一个软件项目来说,并不一定经历所有的测试过程,也不必使用所有的测试方法。行业、用户、时间、经费、功能要求等的差异决定软件项目之间的测试很少有可比性。任何一个软件项目都不能盲目照搬其他软件项目的测试过程,但测试过程中的确应该多取他人之长,多参考一些典型的测试实例。储备了一定的理论知识,再来借"他山之石",并从实践中积累自己的经验,测试就不足为惧了。

本章将在总结和巩固本书前面基本知识的基础上,以网上书店系统为例,重点描述网上书店的集成测试,主要在功能测试上,也有必要的性能测试。后面依次给出了网上书店集成测试阶段的测试计划、测试用例、缺陷(错误)报告、测试结果总结与分析等内容。测试用例将针对网上书店的客户端来设计。该模块不但包含了对数据库的应用,还对系统的并发性、安全性、准确性、高效性都有很高的要求,可谓麻雀虽小,五脏俱全,适合将其进行剖析[8]。

11.1.1 被测试软件项目的背景

大部分读者都有过在网上购物的经历。在网上书店可以很方便地注册、浏览商品、查询商品、购买商品。本章的网上书店实现了上述的基本功能,用户可以在网络书店中进行注册、浏览商品以及查询购物车。

11.1.2 客户端系统介绍

网上书店主页面如图11-1所示。

11.1.3 客户端系统功能需求分析

前台客户功能如图11-2所示。

1. 注册新用户

个人用户可以通过快捷的入口,自愿注册成为网站的会员。个人注册信息中,用户

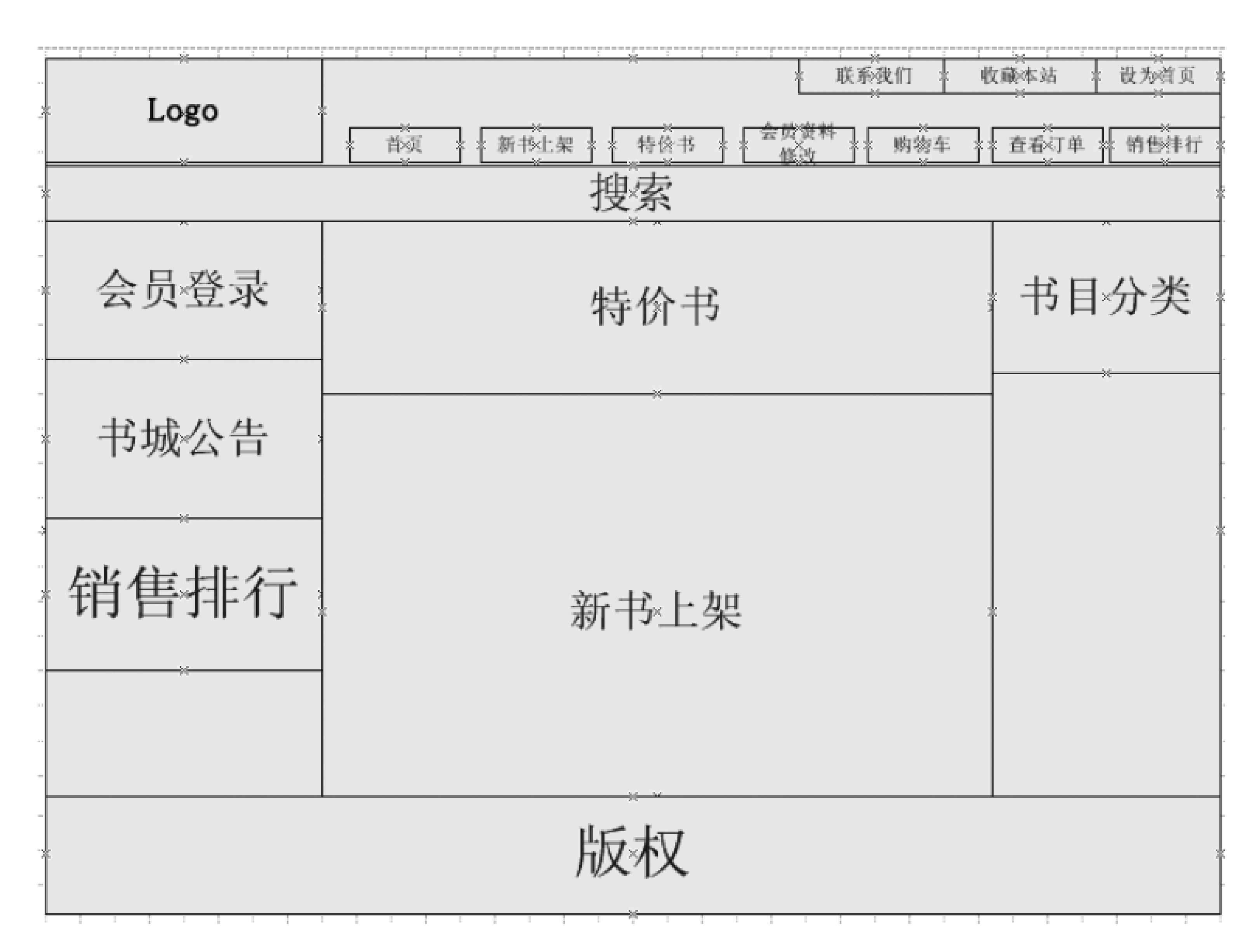


图 11-1　网上书店主页面

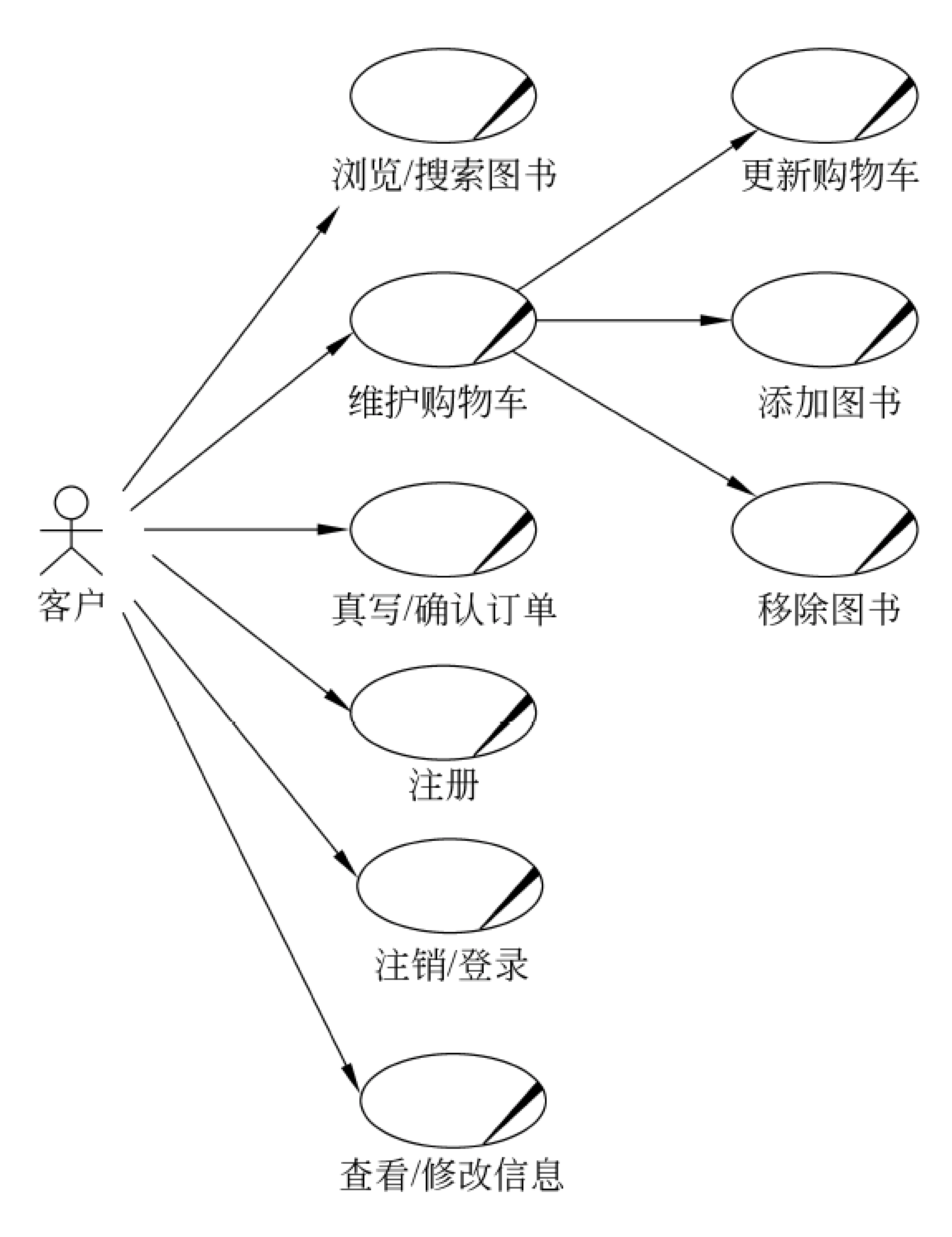


图 11-2　前台客户功能

名和电子邮件地址不允许重复。

注册成功后，系统返回成功祝贺页面，并给其注册邮箱发送 E-mail 确认。

2. 登录/注销系统

客户可以利用快捷入口，或在访问其他受限资源时候，通过填入用户名、密码，被系

统识别为有效的网站用户身份。

同时，为了保护用户安全，防止被机器暴力破解到用户密码，使用验证码技术，在登录时要同时输入验证码，确认是用户人工输入提交的。

客户在完成网站的活动后或者更换用户时可以注销用户。

3. 查看/修改客户信息

客户登录以后，可以预览并修改其个人信息内容。其中，个人信息的一部分已经在注册时填写完毕，其他部分可以陆续添加、修改。

4. 浏览/搜索图书信息

系统将图书信息列出，包括小说、专业、军事图书等信息供所有用户浏览/搜索。

5. 维护购物车

顾客在购买图书后，可以直接将图书添加到购物车中，也可以将错选的图书从购物车中删除，还可以在购物车中查看预已买的商品，以防买错。

6. 更新购物车

购物车会自动在添加图书和移除图书后更新。

7. 添加图书

客户可以将想要购买的图书放到购物车中。

8. 移除图书

客户可以将购物车中不想购买的图书移除。

11.2 项目测试计划

测试计划一般由测试项目经理来制定。测试计划涉及许多测试工作的具体规划，仅有预算、人员安排和时间进度远远不够。一个没有经过很好策划的测试项目不可能顺利开展。

测试工作的成果是提交一份完整的测试计划报告，一般包括被测试项目的背景、测试目标、测试范围、方式、资源、进度安排、测试人员组成以及测试有关的风险等方面。

11.2.1 概述

本次测试的项目是网上书店系统中的客户端功能，即客户的登录端。

本次测试的目的是测试网上书店系统客户端的会员注册登录、在线购买、书目查询等基本功能。所有的用户都可登录网上书店注册购买图书。

本测试计划面向相关项目管理人员、测试人员和开发人员。

11.22 定义

质量风险：被测试系统不能实现描述的产品需求或系统不能达到用户期望的行为，即系统可能存在的错误。

测试用例：为了查找被测试软件中的错误而设计的一系列的操作数据和执行步骤，即一系列测试条件的组合。

测试工具：应用于测试用例的硬件/软件系统，用于安装或撤销测试环境、创造测试条件，执行测试；或者度量测试结果等工作。测试工具独立于测试用例本身。

进入标准：一套决策的指导方针，用于决定项目是否准备好进入特定的测试阶段。在集成测试和系统测试阶段，进入标准会很苛刻。

退出标准：一套标准，用于决定项目是否可以退出当前的测试阶段，或者进入下一个测试阶段或者结束项目。与进入标准相似，测试过程的后几个阶段退出标准一般很苛刻。

功能测试：集中于功能正确性方面的测试。功能测试必须和其他测试方法一起处理潜在的重要的质量风险，如性能、负荷、容积和容量等[8]。

11.23 测试进度计划

测试进度计划包括测试计划、测试系统开发与配置、测试执行和测试总结分析几个阶段，其具体的任务及需提交结果等见表 11-11。

表 11-1 测试进度计划表

阶段	任务号	任务名称	前序任务号	工时	提 交 结 果
测试计划	1	制订测试计划		3	测试计划
测试系统开发与配置	2	人员安排	1	0.5	任务分配
	3	测试环境配置	1,2	3	可运行系统的环境
	4	测试用例设计	1,2	10	测试用例
测试执行	5	第 1 阶段测试通过	1,2,3,4	20	测试结果记录
	6	第 2 阶段测试通过	5	10	测试结果记录
	7	第 3 阶段测试通过	6	5	测试结果记录
测试总结分析	8	退出系统测试	7	3	测试分析报告

11.24 进入标准

（1）“测试小组”配置好软硬件环境，并且可以正确访问这些环境。

（2）“开发小组”已完成所有特性和错误修复并完成修复后的单元测试。

（3）“测试小组”完成“冒烟测试”，程序包能打开，随机的测试操作能正确完成。

11.25 退出标准

（1）开发小组修改完成了所有必须修复的错误。

（2）测试中发现的缺陷按照严重程度分为 4 个级别，如表 11-2 所示。级别不同，严

重程度也不同。“测试小组”执行完所有系统测试的功能、性能测试用例,无 2 级以上遗留问题。如果进行系统测试时,存在严重的质量问题,导致无法继续,并且在可接受的时间范围内无法修复,系统测试终止。

表 11-2 缺陷严重级别

严重级别	严重程度
1—提示(Low)	• 微小的错误,不会影响系统的功能 • 不准确或容易误解的行为和语句
2——般(Medium)	• 该问题增加了测试或用户操作的复杂度 • 该问题轻微降低了系统的性能,但系统仍然能工作
3—严重(High)	• 该问题会严重降低系统的性能 • 不符合客户端需求说明
4—致命(Very High)	• 系统不能正常启动或启动后无法正常工作

(3)“项目管理小组”认为产品实现了稳定性和可靠性。

11.26 测试环境配置

系统的测试环境如图 11-3 所示,系统包括运行在同一台机器上的 20 个虚拟用户和控制器、Web 服务器和数据库以及网络。

系统配置如下:

图 11-3 系统测试环境

1. 软件配置

- 操作系统:Microsoft Windows 2000 Professional、Microsoft Windows 2000 Server、操作系统上必须安装 TOMCAT 6.0 以上版本。
- 数据库系统:Microsoft SQL Server 2000。
- 浏览器:Microsoft IE 6.0 或以上版本。

2. 硬件需求

- CPU:P3 以上。
- 内存:256M 以上。
- 硬盘:20G 以上。

11.27 测试开发

设计测试用例以进行手工测试。

(1) 测试方法:由于本次测试的依据是需求,所以采用黑盒测试方法。

(2) 测试策略:功能测试,主要采用等价类划分的策略。

(3) 测试手段:功能测试,手动模拟正常、异常输入。

(4) 测试内容:功能测试,按照需求功能。

设计开发问题记录及交互工具，包括问题存取控制系统及所对应的数据库，以对测试结果做很好的记录并提供相关测试和开发人员的交互平台。

11.2.8 关键参与者

关键参与者包括具有各自职责的测试经理、测试工程师、测试系统管理员和测试人员等角色，如表 11-3 所示。

表 11-3 关键参与者

角色	小组成员	职责
测试经理	黄志雄	制定测试计划，组织测试工作 系统测试用例评审、测试总结报告评审 提交测试输出文档
测试工程师	郑奇	系统测试案例编写 系统测试案例执行 填写测试跟踪结果报告 系统测试总结报告编写
测试系统管理员	李君	测试环境的搭建 测试软件的维护 测试数据的建立
测试人员	张亮、洪武、毛莹、宋平	负责相关子系统的测试

11.3 测试过程

广义地说，测试工作贯穿一个软件项目开发过程的始终，从项目的策划和相关文档生成开始直到软件通过用户的验收。通常所说的测试是指运行软件系统（或单个的模块）以检验其是否满足用户要求的过程。

网上书店客户端测试按照一般测试过程，将其分为单元测试、集成测试、系统测试、验收测试四个阶段。对于测试开发人员来说，关注的是前三个阶段的测试过程，因此本节详细描述前三个阶段的测试过程。测试过程如图 11-4 所示。

11.3.1 单元测试

单元测试又叫模块测试，是对源程序中每一个程序单元进行测试，检查各个模块是否正确实现了规定的功能，从而发现模块在编码或算法中的错误。该阶段涉及编码和详细设计的文档，由系统开发人员自己来承担。单元测试应对模块内所有重要的控制路径设计测试用例，以便发现模块内部的错误。

根据设计信息选取测试数据，以增大发现各类错误的可能性。在确定测试用例的同时，应给出期望结果。在实际工程项目的开发中，由于开发人员的主要精力集中在系统开发上，在单元测试阶段常常没有时间去做精心的测试用例设计，那么开发人员至少应

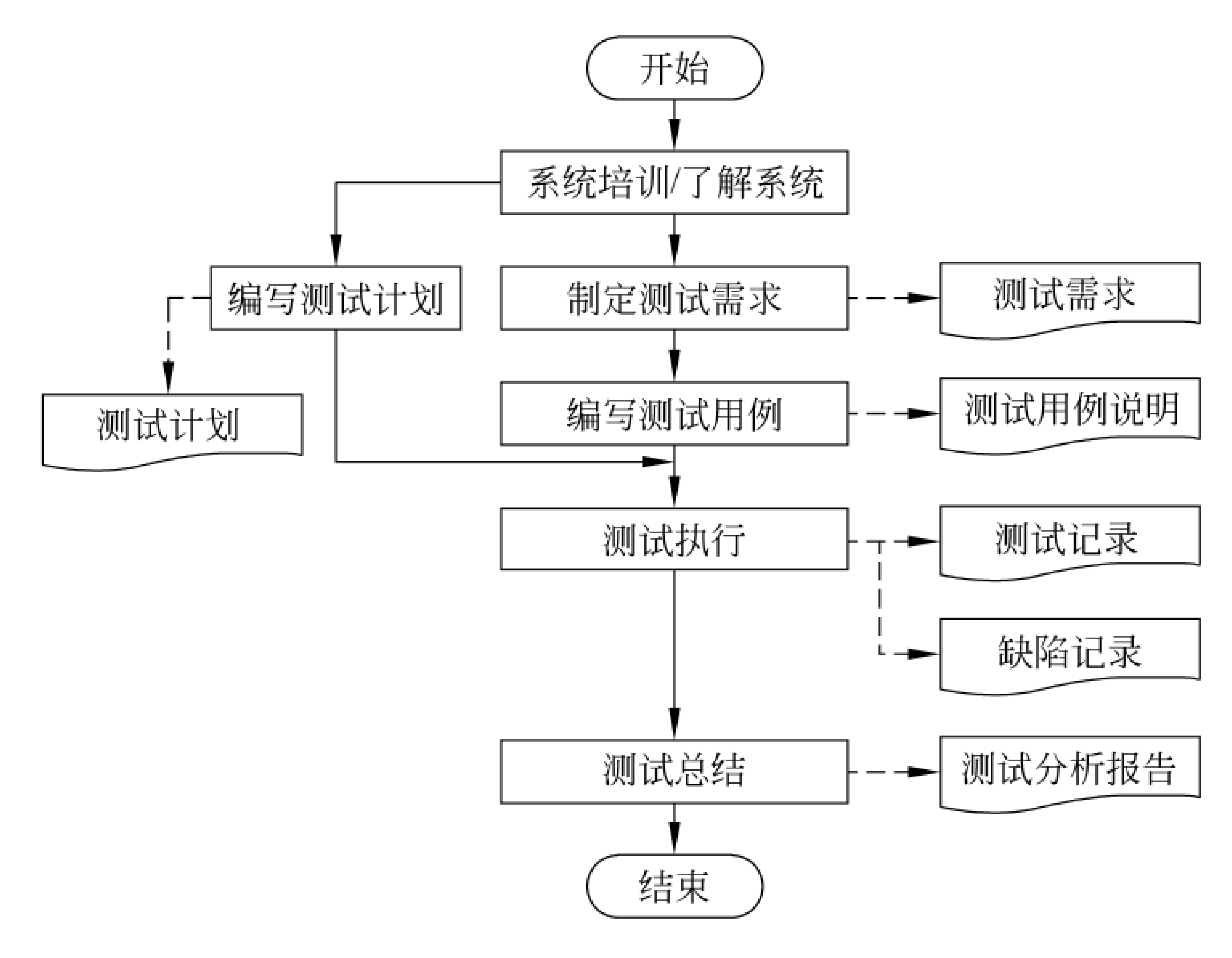


图 11-4 测试过程

该有思路清晰的测试构思和测试大纲。

单元测试常常是动态测试和静态测试两种方式并举的。动态测试可由开发人员去运行局部功能或模块以发现系统潜藏的错误,也可以借助测试工具去测试。静态测试即是代码审查。审查的内容包括代码规则和风格、程序设计和结构、业务逻辑等。

11.3.2 集成测试

集成测试是指将各模块组装起来进行测试,以检查与设计相关的软件体系结构的有关问题,并确认软件是否满足需求规格说明书中确定的各种需求。

网上书店客户端系统的集成测试是指开发人员完成了所有系统模块的开发并通过了单元测试后,将编译好的软件交付给测试部门进行测试的过程。

这个阶段的测试需要一个完备的测试管理过程。集成测试过程可以分为测试准备、测试计划、测试设计、测试执行和测试总结五个阶段。

(1) 测试准备阶段是指测试人员准备测试资源,熟悉系统的过程。

(2) 测试计划阶段包含制定测试策略、资源分配、风险预警和进度安排等内容。此项工作由测试负责人来做。测试计划的模板各不相同,取决于软件的特殊性和管理的规范性。

(3) 测试设计阶段包括设计测试用例及相关管理工具的设计。网上书店客户端集成测试过程中的主要测试用例,侧重于系统的功能和性能测试,将在 11.4 节详细介绍。测试用例设计之前一般要有一个测试用例的设计大纲。

(4) 完成测试设计工作后,就开始执行实际的测试工作。

测试时另外一项非常重要的工作就是做好系统缺陷记录。网上书店客户端系统生成缺陷报告的注意事项以及缺陷报告的实例参见 11.5 节。

经过修改后的系统再次经过测试即是回归测试。

（5）测试结束后要及时总结分析测试结果。测试结果的总结与分析，一方面是提供一个系统功能、性能和稳定性等方面的完整的分析和结论；另一方面要对测试过程本身做出总结，总结成功的经验和失败的教训，以便日后的工作开展得更顺利。

11.3.3 系统测试

系统测试是在真实或模拟系统运行的环境下，检查完整的程序系统能否和系统（包括硬件、外设、网络和系统软件、支持平台等）正确配置、连接，并满足用户需求。

系统测试也应该经过测试准备、测试计划、测试设计、测试执行和测试总结五个阶段。每个阶段所做工作内容与集成测试很相似，只是关注点有所不同。

在网上书店客户端的系统测试中，要搭建更真实的运行环境，另外还要在不同的操作系统下进行测试，如数据库服务器分别搭建在 UNIX 和 Windows NT 环境下，进行长时间多客户端并发运行系统的各项功能测试，并观测服务器的承受能力（系统的反应时间、服务器的资源占用情况等）。

11.3.4 验收测试

验收测试是指在用户对软件系统验收之前组织的系统测试。测试人员都是真正的用户，在尽可能真实的环境下进行操作，并将测试结果进行汇总，由相关管理人员对软件做出评价以及是否验收的决定。

网上书店客户端一般在用户验收之前都需要对系统进行一段时间的试运行，因此可以说网上书店客户端的验收测试就是实际的使用（但用户一般要参与软件的系统测试，即所谓的 β 测试，不然用户是不会放心让系统试运行的）。

因为验收测试由用户完成，不同软件实际应用的差异性又很大，这里就不对其详加论述了。

11.4 测试用例设计

测试用例应由测试人员在充分了解系统的基础上，在测试之前设计好。测试用例的设计是测试系统开发中一项非常重要的内容。集成测试阶段测试用例的设计依据为系统需求分析、系统用户手册和系统设计报告等相关资料的内容，而且测试人员要与开发人员充分交互。另外，有一些内容由测试人员的相关背景知识、经验、直觉等产生。

测试用例的设计需要考虑很周全。在测试系统功能的同时，还要检查系统对输入数据（合法值、非法值和边界值）的反应，要检查合法的操作和非法的操作，检查系统对条件组合的反应等。好的测试用例让其他人能够很好地执行测试，能够快速地遍历所测试的功能，能够发现至今没有发现的错误。所以测试用例应该由经验丰富的系统测试人员来编写，对于新手来说，应该多阅读一些好的测试用例，并且在测试实践中用心去体会。

在编写测试用例之前，应该给出测试大纲。大纲基本上是测试思路的整理，以保证测试用例的设计能够清晰、完整而不是顾此失彼。测试大纲可以按照模块、功能点、菜单

和业务流程这样的思路来策划。

本节给出“网上书店系统”的“客户端子系统”的测试用例。

11.4.1　测试覆盖设计

由于本次测试是系统测试，测试的依据是系统需求，测试的设计应该满足对需求的覆盖，所以，采用的测试方法主要是黑盒测试，包括等价类划分（有效测试和无效测试）、边界值和错误猜测法等。测试用例覆盖矩阵如表 11-4 所示。

表 11-4　测试用例功能/性能覆盖矩阵

序号	功　能　项	测 试 用 例	优先级
01	所有基本页面的链接正确	TestCase-FUNC-01	中
02	所有页面的转移正确	TestCase-FUNC-02	中
03	会员信息列表正确	TestCase-FUNC-03	高
04	图书详细信息正确	TestCase-FUNC-04	高
05	正常购买图书的流程	TestCase-FUNC-05	高
06	查询图书的流程	TestCase-FUNC-06	高
07	会员登录、退出的流程	TestCase-FUNC-07	高

11.4.2　功能测试用例

按照表 11-4，设计相应的测试用例如下。

1. 基本页面链接的测试用例

测试编号是 TestCase-FUNC-01，测试内容是验证所有基本页面链接的正确性，同时所有的页面都按照需求有正确的显示。这个测试用例的具体设计如表 11-5 所示。

表 11-5　TestCase-FUNC-01 测试用例

测试项目名称：网上书店系统—客户端	
试用例编号：TestCase-FUNC-01 **测试项目标题**：所有基本页面的正确链接	**测试人员**：小组人员 **测试时间**：2008/9/3
测试内容： 验证网站首页所有链接有效 验证网站首页中图片能正确装入 验证网站首页中的超连接的连接页面与页面中指示（或 图示）相符 验证网站中各个页面的显示的信息都符合需求 **测试环境与系统配置**： 详见《测试计划》 **测试输入**：脚本见 TC-F-01. c（见测试开发文档） **数据**： **测试次数**：每个测试过程做 2 次	

续表

测试项目名称：网上书店系统—客户端
预期结果： 可以正确显示图片，每个链接有效，超连接的连接页面与页面中指示(或 图示)相符 **测试过程：** 登录 http://www.qiqishop.com 对于首页每个链接，点击进入。察看链接的页面是否相符合 对于首页每个链接，点击查看能否进入相应页 **测试结果：** **测试结论：** **实现限制：** **备注：**

2. 页面转移正确性的测试用例

测试编号是 TestCase-FUNC-02，测试内容是测试所有转移页面链接的正确性，同时所有的页面都按照需求有正确的显示。这个测试用例的具体设计如表 11-6 所示。

表 11-6 TestCase-FUNC-02 测试用例

测试项目名称：网上书店系统—客户端	
试用例编号：TestCase-FUNC-02	**测试人员：**小组人员 **测试时间：**2008/9/3
测试项目标题：转移页面的正确性 **测试内容：** 验证网站每页输入“转到 ”的输入框能正确处理输入 **测试环境与系统配置：** 详见《测试计划》 **测试输入：**异常数据：0，1.4，6 **数据：**　　正常数据：1 **测试次数：**每个测试过程做 2 次 **预期结果：** 对于正常数据能够转到相应页面，异常数据能够报错 **测试过程：** 登录 http://www.qiqishop.com 对于首页“转到”的输入框，依次输入如上数据 **测试结果：** **测试结论：** **实现限制：** **备注：**	

3. 显示图书列表的测试用例

测试编号是 TestCase-FUNC-03，测试内容是测试所有职位列表页面的正确性，同时所有的页面都按照需求有正确的显示。这个测试用例的具体设计如表 11-7 所示。

表 11-7 TestCase-FUNC-03 测试用例

测试项目名称：网上书店系统—客户端

测试用例编号：TestCase-FUNC-03　　**测试人员**：小组成员
测试时间：2008/9/3

测试项目标题：职位列表的显示

测试内容：
- 验证网页上的表格是否正确显示
- 验证在图书列表中是否正确显示图书名称、图书出版日期、图书类型、图书数量
- 验证图书列表是否按职位添加日期排序

测试环境与系统配置：
软件环境：Micrsoft Windows XP Professional
硬件环境：P4 1.7GHz CPU+1.7GHz 512MB 内存
网络环境：5 人共享 1Mbps 带宽
测试输入：无
数据：
测试次数：
应在至少 2 种浏览器中进行测试，并刷新 2 次。
预期结果：
网页正确显示，在图书列表中显示图书名称、图书出版日期、图书类型、图书数量，按添加时间排序
测试过程：
在 IE 浏览器地址栏中输入 http://www.qiqishop.com，并刷新
在遨游浏览器地址栏中输入 http://www.qiqishop.com，并刷新
测试结果：
测试结论：
实现限制：无
备注：无

4. 图书购买流程的测试用例

测试编号是 TestCase-FUNC-04，测试内容是测试所有职位详细信息页面的正确性，同时所有的页面都按照需求有正确的显示。这个测试用例的具体设计如表 11-8 所示。

表 11-8 TestCase-FUNC-04 测试用例

测试项目名称：网上书店系统—客户端

测试用例编号：TestCase-FUNC-04　　**测试人员**：小组人员
测试时间：2008/9/3

测试项目标题：图书购买流程

续表

测试项目名称：网上书店系统—客户端
测试内容： 验证页面是否正确显示了职位名称、职位描述、职位要求和招聘人数几项 验证职位详细信息页面上的信息是否与职位列表中有关的信息相符 **测试环境与系统配置：** 软件环境：Micrsoft Windows XP Professional+Micrsoft IE 6.0 硬件环境：P4 1.7GHz CPU+512MB 内存 网络环境：6 人共享 1Mbps 带宽 **测试输入：**无 **数据：** **测试次数：** 应至少测试 3 次购买过程 **预期结果：** 购买成功 **测试过程：** 购买一至多本图书。 **测试结果：** **测试结论：** **实现限制：**无 **备注：**无

5. 会员登录的测试用例

测试编号是 TestCase-FUNC-05，测试内容是测试会员在正常（非正常）输入的条件下是否可以将信息成功提交，同时所有的页面都按照需求有正确的显示。这个测试用例的具体设计如表 11-9 所示。

表 11-9　TestCase-FUNC-05 测试用例

测试项目名称：网上书店系统—客户端	
测试用例编号：TestCase-FUNC-05	**测试人员：**小组人员 **测试时间：**2008/9/3
测试项目标题：正常登录流程的功能测试	
测试内容： • 输入账号 • 输入密码 • 点击登录 **测试环境与系统配置：** 软件环境：Micrsoft Windows XP Professional+Micrsoft IE 6.0 硬件环境：P4 2.8GHz CPU+2.79GHz 512MB 内存 网络环境：5 人共享 1Mbps 带宽 详见《测试计划》 **测试输入：**脚本 TC-F-02.c（见测试开发文档） **数据：**	

续表

测试项目名称：网上书店系统—客户端
测试次数：每个测试过程做 2 次 **预期结果**： • 不填写账号时提示出错 • 不填写密码时提示出错 • 填写错误账号和密码时提示出错 **测试过程**： 登录 http://www.qiqishop.com 在文本框中分别输入账号和密码 单击“提交”按钮 提交并返回 **测试结果**： **测试结论**： **实现限制**： **备注**：

6. 图书查询的测试用例

测试编号是 TestCase-FUNC-06，测试内容是测试查询图书在非正常输入时系统的异常处理，同时所有的页面都按照需求有正确的显示。这个测试用例的具体设计如表 11-10 所示。

表 11-10　TestCase-FUNC-06 测试用例

测试项目名称：网上书店系统—客户端	
测试用例编号：TestCase-FUNC-06	**测试人员**：小组人员 **测试时间**：2008/9/2
测试项目标题：基本信息页面的功能测试 **测试内容**： 对于基本测试页面，测试其对异常数据的处理 **测试环境与系统配置**： 详见《测试计划》 **测试输入**：留空 **数据**：　正确图书 　　　　错误图书 **测试次数**：每个测试过程做 2 次 **预期结果**： 出现预期结果 **测试过程**： 登录 http://www.qiqishop.com 在搜索框中输入查询图书。 单击“搜索”按钮 输入异常数据(数据如上所示)	

续表

测试项目名称：网上书店系统—客户端
提交 **测试结果：** **测试结论：** **实现限制：** **备注：**

11.5 测试报告和分析

11.5.1 缺陷报告

测试过程缺陷数据的总结情况如表 11-11 所示。从缺陷的严重程度看，严重和致命的缺陷占的比例较大，说明系统还是存在严重问题，需要重新修改，产品不能提交。从缺陷的类型看，功能缺陷、赋值缺陷和设计缺陷占比较大的比例，说明设计和编码过程中存在很大的问题。

表 11-11 测试过程的缺陷数据

被测对象	总数	严 重 程 度				缺 陷 类 型					
		致命	严重	一般	提示	设计错误	赋值错误	算法错误	接口错误	功能错误	其他
首页	3	0	1	0	0	2	0	0	0	1	0
整个系统	1	0	1	0	0	0	0	0	0	9	0

根据缺陷跟踪结果得出缺陷的严重程序分布和的缺陷的类型分布分别如图 11-5 和图 11-6 所示。

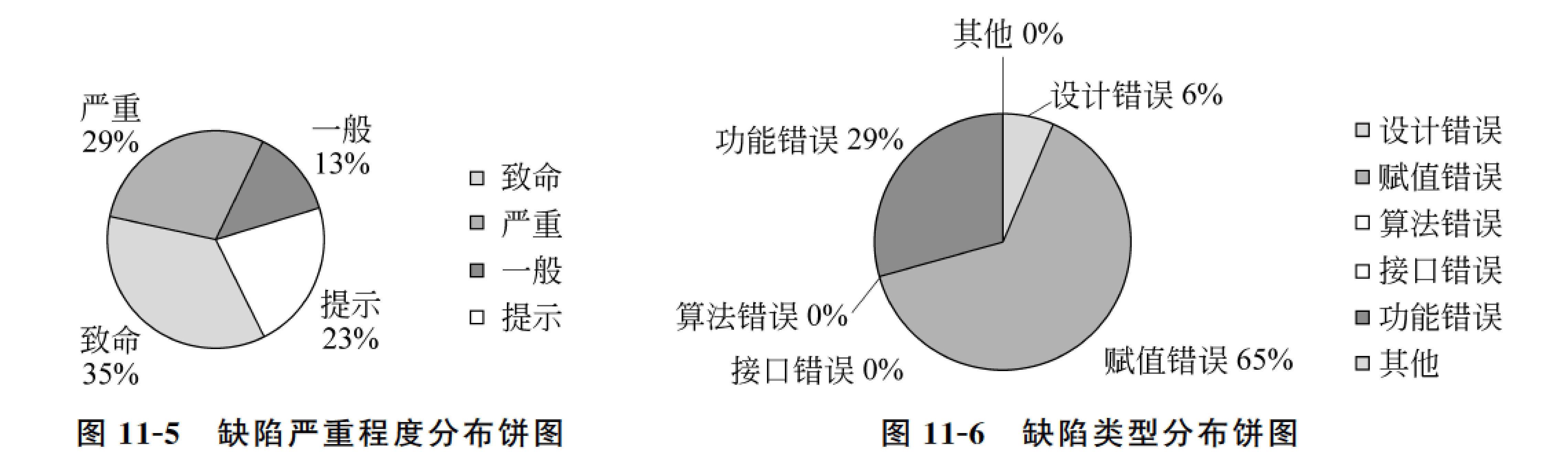


图 11-5 缺陷严重程度分布饼图 **图 11-6 缺陷类型分布饼图**

11.5.2 测试总结报告

测试总结报告的模板如表 11-12 所示。各行业、各阶段的软件测试会有具体不同的总结报告，但基本上应该有本模板所展示的项目。

表 11-12 测试总结报告模板

***测试报告		
项目编号：	项目名称：	
项目软件经理：	测试负责人：	
测试时间：		
测试目的与范围：		
	测试环境	
名称：		软件版本：
服务器操作系统：		
数据库：		
应用服务器：		
测试软件：		
测试机操作系统：		
测试数据说明：		
总体分析：		
典型性具体测试结果：		

11.5.3 测试用例分析

对工作的及时总结，会及时调整方向，大大提高工作效率。测试工作的效果要直接依赖测试用例的编写和执行状况，所以在测试过程中和测试结束后都要对关于测试用例的一些重要值进行度量。

关于测试用例的分析，通常包括以下的内容：

- 计划了多少个测试用例，实际运行了多少个？
- 有多少测试用例失败了？
- 在失败的测试用例中，有多少个在错误得到修改后最终运行成功了？
- 这些测试平均占用的运行时间比预期的长还是短？
- 有没有跳过一些测试，如果有，原因是什么？
- 测试有没有覆盖了所有影响系统性能的重要事件？

这些问题都可以从相关的测试用例的设计和测试问题记录中找到相应的答案。当然，如果使用了数据库，这些问题就更能轻松地被解答了。测试用例的分析报告可以以文字描述、表、图等多种形式体现出来。

11.5.4 软件测试结果统计分析

在对软件产品测试过程中发现的问题进行充分分析、归纳和总结的基础上，由全体参与测试的人员完成“软件问题倾向分析表”，对该软件或该类型系统软件产品在模块、功能及操作等方面出错倾向及其主要原因进行分析。软件问题倾向分析表将为以后开发工作提供一个参考，使开发人员根据软件问题倾向分析表明确在开发过程中应注意和回避的问题。该表也可为以后的测试工作明确测试重点提供依据。

软件的不同版本与测试时检测出的缺陷数的对应关系如图 11-7 所示。图中的版本指的是同一软件经过不同的测试阶段并修复缺陷及作必要的调整后所产生的软件产品。显然，该图所表达的测试结果的变化是非常理想的。

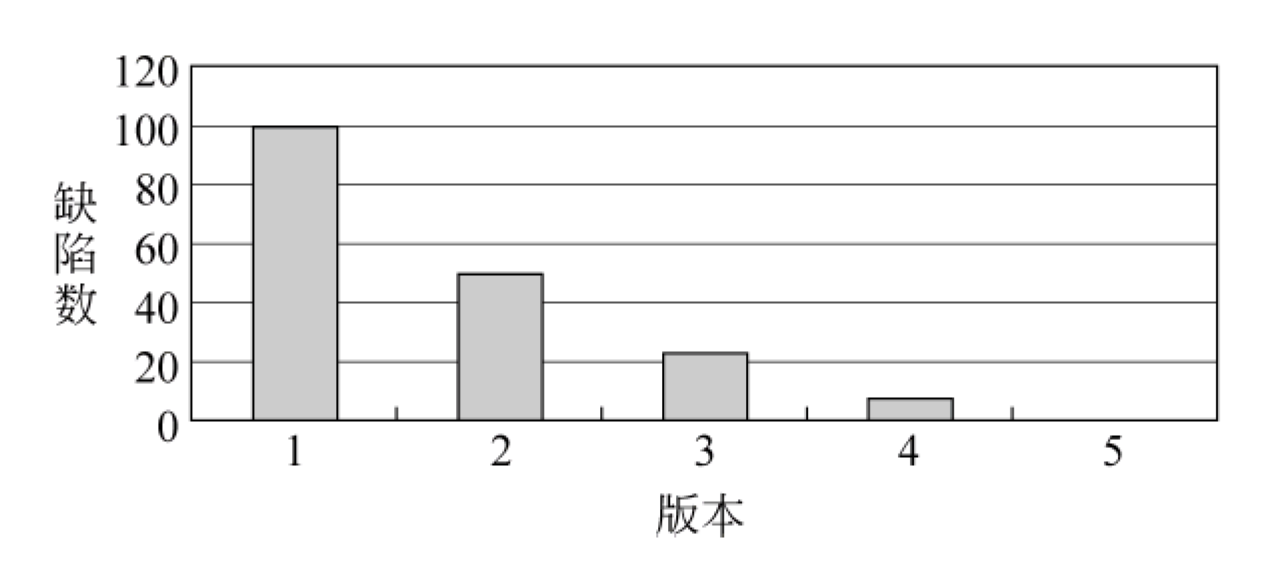


图 11-7　按版本统计结果示例

一个测试阶段所发现的缺陷数与测试日期之间的对应关系如图 11-8 所示。测试过程中所发现的缺陷是随着时间的推移而增多的，但一段时间后，测试所发现的缺陷增加会渐缓，甚至没有增加。如果测试还在进行，那么表明，在现有测试用例、软硬件环境及相关条件下已经很难再发现新的缺陷了(即使可以肯定系统中仍然存在缺陷)，那么这个测试阶段应该考虑停止了。

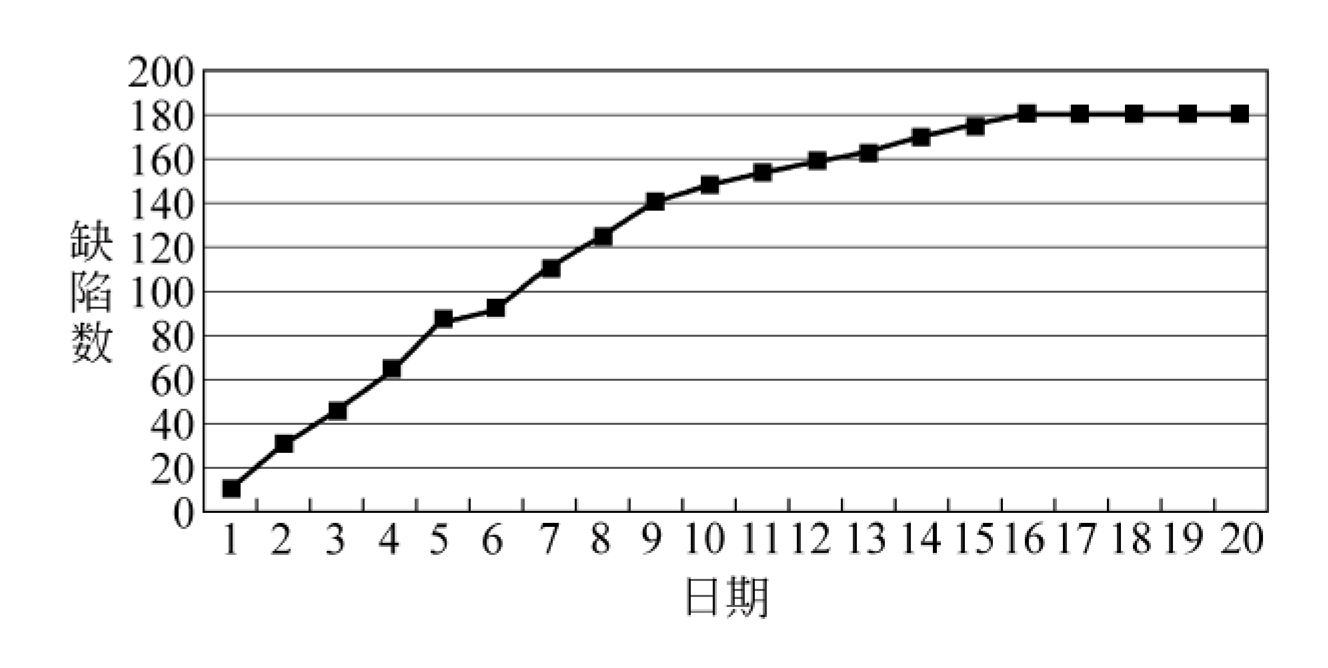


图 11-8　按日期统计结果示例

测试所发现的缺陷数目与究其原因缺陷所属的软件工程的不同阶段之间的关系如图 11-9 所示。这个图表会又一次验证软件工程的任何阶段都会有导致程序中产生错误的因素，只是程度和数目不同而已。通过该图表的分析，可以清楚地看到，软件工程中的哪个阶段更应该加强控制。

实际的测试结果总结分析还有很多情形，这里列出的是一些比较典型的分析图表。

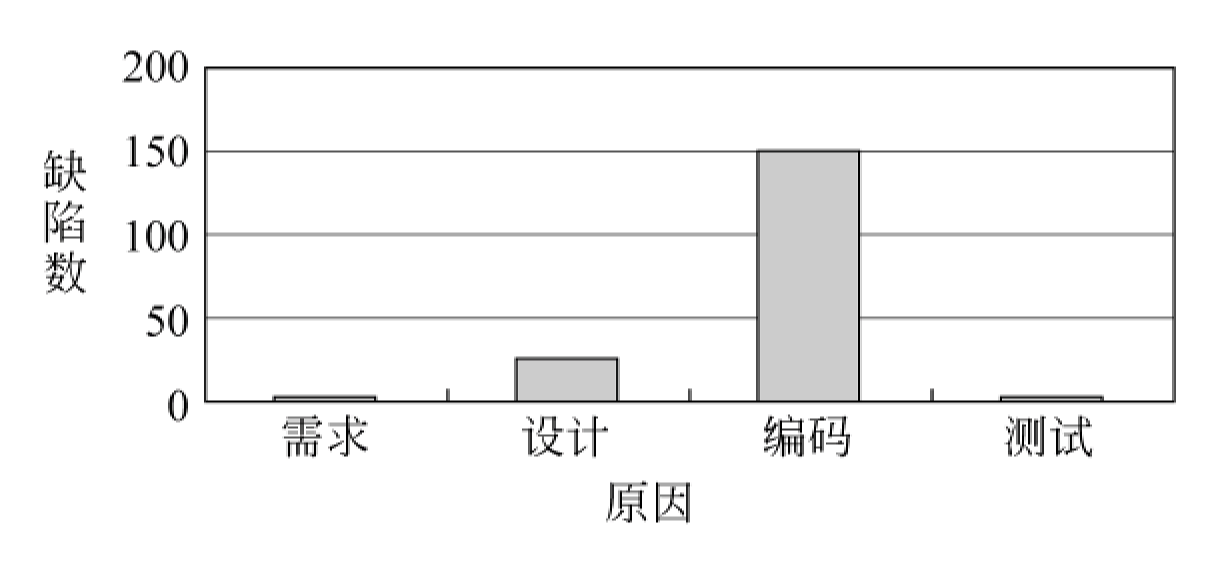


图 11-9 按原因统计结果示例

实际工作的不同需要会有不同选择。而这些分析数据、图表是与测试结果分析报告配合使用的。

附录 A



术　语　表

Acceptance Testing——可接受性测试：一般由用户/客户进行的确认是否可以接受一个产品的验证性测试。

Ad Hoc Testing——随机测试：测试人员通过随机的尝试系统的功能，试图使系统中断。

Alpha Testing——Alpha 测试：由选定的用户进行的产品早期性测试。这个测试一般在可控制的环境下进行。

anomaly——异常：在文档或软件操作中观察到的任何与期望违背的结果。

architecTrue——构架：一个系统或组件的组织结构。

ASQ——自动化软件质量(Automated Software Quality)：使用软件工具来提高软件的质量。

assertion——断言：指定一个程序必须已经存在的状态的一个逻辑表达式，或者一组程序变量在程序执行期间的某个点上必须满足的条件。

assertion checking——断言检查：用户在程序中嵌入的断言的检查。

audit——审计：一个或一组工作产品的独立检查以评价与规格、标准、契约或其他准则的符合程度。

audit trail——审计跟踪：系统审计活动的一个时间记录。

Automated Testing——自动化测试：使用自动化测试工具来进行测试，这类测试一般不需要人干预，通常在 GUI、性能等测试中用得较多。

basis test set——基本测试集：根据代码逻辑引出来的一个测试用例集合，它保证能获得 100%的分支覆盖。

Beta Testing——Beta 测试：在客户场地，由客户进行的对产品预发布版本的测试。这个测试一般是不可控的。

big-bang testing——大锤测试/一次性集成测试：非渐增式集成测试的一种策略，测试的时候把所有系统的组件一次性组合成系统进行测试。

Black Box Testing——黑盒测试：根据软件的规格对软件进行的测试，这类测试不考虑软件内部的运作原理，因此软件对用户来说就像一个黑盒子。

bottom-up testing——由低向上测试：渐增式集成测试的一种，其策略是先测试底层的组件，然后逐步加入较高层次的组件进行测试，直到系统所有组件都加入到系统。

boundary value testing——边界值测试：通过边界值分析方法来生成测试用例的一种测试策略。

branch condition combination testing——分支条件组合测试：通过执行分支条件结果组合来设计测试用例的一种方法。

branch condition testing——分支条件测试：通过执行分支条件结果来设计测试用例的一种方法。

branch testing——分支测试：通过执行分支结果来设计测试用例的一种方法。

Breadth Testing——广度测试：在测试中测试一个产品的所有功能，但是不测试更细节的特性。

CAST——计算机辅助测试：在测试过程中使用计算机软件工具进行辅助的测试。

code audit ——代码审计：由一个人、组或工具对源代码进行的一个独立的评审，以验证其与设计规格、程序标准的一致性。正确性和有效性也会被评价。

Code Inspection——代码检视：一个正式的同行评审手段，在该评审中，作者的同行根据检查表对程序的逻辑进行提问，并检查其与编码规范的一致性。

code-based testing——基于代码的测试：根据从实现中引出的目标设计测试用例。

Compatibility Testing——兼容性测试：测试软件是否和系统的其他与之交互的元素之间兼容，如浏览器、操作系统、硬件等。

conformance criterion——一致性标准：判断组件在一个特定输入值上的行为是否符合规格的一种方法。

Conformance Testing——一致性测试：测试一个系统的实现是否和其基于的规格相一致的测试。

conversion testing——转换测试：用于测试已有系统的数据是否能够转换到替代系统上的一种测试。

data definition-use testing——数据定义使用测试：以执行数据定义使用对为目标进行测试用例设计的一种技术。

data flow testing——数据流测试：根据代码中变量的使用情况进行的测试。

dead code——死代码：在程序操作过程中永远不可能被执行到的代码。

Debugging——调试：发现和去除软件失效根源的过程。

Depth Testing——深度测试：执行一个产品的一个特性的所有细节，但不测试所有特性。比较广度测试。

dirty testing——肮脏测试：参考负面测试(negative testing)。

documentation testing——文档测试：测试关注于文档的正确性。

dynamic analysis——动态分析：根据执行的行为评价一个系统或组件的过程。

Dynamic Testing——动态测试：通过执行软件的手段来测试软件。

End-to-End testing——端到端测试：在一个模拟现实使用的场景下测试一个完整的应用环境，例如和数据库交互，使用网络通信等。

equivalence partition testing——等价划分测试：根据等价类设计测试用例的一种技术。

Exhaustive Testing——穷尽测试：测试覆盖软件的所有输入和条件组合。

FMEA——失效模型效果分析(Failure Modes and Effects Analysis)：可靠性分析中的一种方法，用于在基本组件级别上确认对系统性能有重大影响的失效。

FTA——故障树分析(Fault Tree Analysis)：引起一个不需要事件产生的条件和因素的确认和分析，通常是严重影响系统性能、经济性、安全性或其他需要特性。

Functional Testing——功能测试：测试一个产品的特性和可操作行为以确定它们满足规格。

glass box testing——玻璃盒测试：参考白盒测试(White Box Testing)。

incremental testing——渐增测试：集成测试的一种，组件逐渐被增加到系统中直到整个系统被集成。

installability testing——可安装性测试：确定系统的安装程序是否正确的测试。

Integration Testing——集成测试：测试一个应用组合后的部分以确保它们的功能在组合之后正确。该测试一般在单元测试之后进行。

interface testing——接口测试：测试系统组件间接口的一种测试。

isolation testing——孤立测试：组件测试(单元测试)策略中的一种，把被测组件从其上下文组件之中孤立出来，通过设计驱动和桩进行测试的一种方法。

job control language——工作控制语言：用于确定工作顺序，描述它们对操作系统要求并控制它们执行的语言。

LCSAJ——线性代码顺序和跳转(Linear Code Sequence And Jump)：包含三个部分：可执行语句线性顺序的起始，线性顺序的结束，在线性顺序结束处控制流跳转的目标语句。

Load Testing——负载测试：通过测试系统在资源超负荷情况下的表现，以发现设计上的错误或验证系统的负载能力。

logic-coverage testing——逻辑覆盖测试：参考结构化测试用例设计(structural test case design)。

maintainability testing——可维护性测试：测试系统是否满足可维护性目标。

modified condition/decision testing——修改条件/判定测试：根据 MC/DC 设计测试用例的一种技术。

Monkey Testing——跳跃式测试：随机性，跳跃式的测试一个系统，以确定一个系统是否会崩溃。

mutation analysis——变体分析：一种确定测试用例套完整性的方法，该方法通过判断测试用例套能够区别程序与其变体之间的程度。

Negative Testing——逆向测试/反向测试/负面测试：测试瞄准于使系统不能工作。

non-functional requirements testing——非功能性需求测试：与功能不相关的需求测试，如性能测试、可用性测试等。

N-switch testing——N 切换测试：根据 N 转换顺序设计测试用例的一种技术，经常用于状态转换测试中。

operational testing——可操作性测试：在系统或组件操作的环境中评价它们的

表现。

partition testing——分类测试：参考等价划分测试(equivalence partition testing)。

path testing——路径测试：根据路径设计测试用例的一种技术，经常用于状态转换测试中。

performance testing——性能测试：评价一个产品或组件与性能需求是否符合的测试。

Positive Testing——正向测试：测试瞄准于显示系统能够正常工作。

precondition——预置条件：环境或状态条件，组件执行之前必须被填充一个特定的输入值。

progressive testing——递进测试：在先前特性回归测试之后对新特性进行测试的一种策略。

pseudo-random——伪随机：看似随机的，实际上是根据预先安排的顺序进行的。

QA——质量保证(quality assurance)：

(1) 已计划的系统性活动，用于保证一个组件、模块或系统遵从已确立的需求。

(2) 采取的所有活动以保证一个开发组织交付的产品满足性能需求和已确立的标准和过程。

QC——质量控制(quality control)：用于获得质量需求的操作技术和过程，如测试活动。

recovery testing——恢复性测试：验证系统从失效中恢复能力的测试。

regression analysis and testing——回归分析和测试：一个软件验证和确认任务以确定在修改后需要重复测试和分析的范围。

Regression Testing——回归测试：在发生修改之后重新测试先前的测试以保证修改的正确性。

requirements-based testing——基于需求的测试：根据软件组件的需求导出测试用例的一种设计方法。

Sanity Testing——理智测试：软件主要功能成分的简单测试以保证它是否能进行基本的测试。参考冒烟测试。

serviceability testing——可服务性测试：参考可维护性测试(maintainability testing)。

Smoke Testing——冒烟测试：对软件主要功能进行快餐式测试。最早来自于硬件测试实践，以确定新的硬件在第一次使用的时候不会着火。

spiral model——螺旋模型：软件开发过程的一个模型，其中的组成活动，典型的包括需求分析，概要设计，详细设计，编码，集成和测试等活动被迭代的执行直到软件被完成。

statement testing——语句测试：根据语句覆盖来设计测试用例的一种方法。

Static Testing——静态测试：不通过执行来测试一个系统。

statistical testing——统计测试：通过使用对输入统计分布进行分析来构造测试用例的一种测试设计方法。

storage testing——存储测试：验证系统是否满足指定存储目标的测试。

Stress Testing——压力测试：在规定的规格条件或者超过规定的规格条件下，测试一个系统，以评价其行为。类似负载测试，通常是性能测试的一部分。

structural testing——结构化测试：参考结构化测试用例设计（structural test case design）。

System Testing——系统测试：从一个系统的整体而不是个体上来测试一个系统，并且该测试关注的是规格，而不是系统内部的逻辑。

technical requirements testing——技术需求测试：参考非功能需求测试（non-functional requirements testing）。

test case——测试用例：用于特定目标而开发的一组输入、预置条件和预期结果。

test case design technique——测试用例设计技术：选择和导出测试用例的技术。

test case suite——测试用例套：对被测软件的一个或多个测试用例的集合。

test comparator——测试比较器：一个测试工具用于比较软件实际测试产生的结果与测试用例预期的结果。

test completion criterion——测试完成标准：一个标准用于确定被计划的测试何时完成。

test driver——测试驱动：一个程序或测试工具用于根据测试套执行软件。

test environment——测试环境：测试运行其上的软件和硬件环境的描述，以及任何其他与被测软件交互的软件，包括驱动和桩。

test execution——测试执行：一个测试用例被被测软件执行，并得到一个结果。

test execution technique——测试执行技术：执行测试用例的技术，包括手工、自动化等。

test generator——测试生成器：根据特定的测试用例产生测试用例的工具。

test harness——测试用具：包含测试驱动和测试比较器的测试工具。

test log——测试日志：一个关于测试执行所有相关细节的时间记录。

test measurement technique——测试度量技术：度量测试覆盖率的技术。

Test Plan——测试计划：一个文档，描述了要进行的测试活动的范围、方法、资源和进度。它确定测试项、被测特性、测试任务、谁执行任务，并且任何风险都要冲突计划。

test procedure——测试规程：一个文档，提供详细的测试用例执行指令。

Test Script——测试脚本：一般指的是一个特定测试的一系列指令，这些指令可以被自动化测试工具执行。

Test Specification——测试规格：一个文档，用于指定一个软件特性、特性组合或所有特性的测试方法、输入、预期结果和执行条件。

test strategy——测试策略：一个简单的高层文档，用于描述测试的大致方法、目标和方向。

test suite——测试套：测试用例和/或测试脚本的一个集合，与一个应用的特定功能或特性相关。

test target——测试目标：一组测试完成标准。

testability——可测试性：一个系统或组件有利于测试标准建立和确定这些标准是否被满足的测试执行的程度。

thread testing——线程测试：自顶向下测试的一个变化版本，其中，递增的组件集成遵循需求子集的实现。

top-down testing——自顶向下测试：集成测试的一种策略，首先测试最顶层的组件，其他组件使用桩，然后逐步加入较低层的组件进行测试，直到所有组件被集成到系统中。

附录 B



IEEE 模板

美国电气电子工程师协会(IEEE)制定软件测试行业标准,规定了软件测试从制定计划到用例设计、实施、结束等各个阶段的基本要求及模板。本附录列出了 IEEE 标准 829-1998 软件测试文档编制标准每个模板的描述。完整的 IEEE 指南可以从 IEEE 的 Web 网站 www.ieee.org 购买下载。

测试文档模板

目录

1. 测试计划

   用于总体测试计划和针对等级的测试计划。

2. 测试设计规格说明

   用于每个测试等级,以指定测试集的体系结构和覆盖跟踪。

3. 测试用例规格说明

   制定符合测试设计说明书中测试条件的测试数据。

4. 测试规程规格说明

   用于指定执行一个测试用例集的步骤。

5. 测试日志

   记录运行了哪些测试用例、执行人、执行顺序以及用例是否通过。

6. 测试意外事件报告

   用来描述出现在测试过程或产品中的异常情况,这些异常情况可能存在于需求、设计、代码、文档或测试用例中。随后,可以将意外事件归类为缺陷或增强事件。

7. 测试总结报告

   管理报告,提供测试完成后未覆盖到的重要信息,包括对测试结果的质量评定,测试下软件系统的质量,以及事件报告的统计资料。为改善未来的测试计划,报告同时记录所完成的测试种类及测试时间。最终文档用来指出软件系统在测试中是否达到了项目关系人的验收标准。

测试计划模板

1. 目录
2. 测试计划标识符
3. 介绍
4. 测试项
5. 软件风险问题
6. 待测特征
7. 不予测试的特征
8. 方法
9. 测试项通过/失败准则
10. 挂起准则和恢复需求
11. 测试交付物
12. 测试任务
13. 环境需求
14. 职责
15. 人员安排与培训需求
16. 进度表
17. 计划风险与应急措施
18. 审批

测试设计规格说明版本

目录

1. 测试设计规格说明标识符
2. 待测特征
3. 方法细化
4. 测试标识
5. 特征通过/失败准则

测试用例规格说明模板

目录

1. 测试用例规格说明标识符
2. 测试项
3. 输入规格说明
4. 输出规格说明
5. 环境需求
6. 特殊规程需求
7. 用例间的相关性

测试规程模板

目录

1. 测试规程规格说明标识符

为这个测试规程制定唯一的标识符。提供一个到相应的测试设计规格说明的引用。

2. 目的

描述规程的目的，并应用到被执行的测试用例中。

3. 特殊需求

描述各种特殊的需求，例如环境需求、技能水平、培训等。

4. 规程步骤

这是测试规程的核心部分。IEEE 描述如下几个步骤：

4.1 记录

描述记录测试执行结果、观察到的意外事件，以及其他与测试相关的事件所用的各种特定方法和格式。

4.2 准备

描述执行这个规程需要准备的一系列活动。

4.3 开始

描述开始执行这个规程需要的各种活动。

4.4 进行

描述在这个规程的执行期间需要的所有活动。

4.5 度量

描述如何进行测试的度量。

4.6 中止

描述发生非计划事件时暂停测试需要采取的活动。

4.7 重新开始

指明规程中各个重新开始的位置，并描述从这个位置重新开始所需的步骤。

4.8 停止

描述正常停止执行所需的各种活动。

4.9 完成

描述恢复环境所需要的活动。

4.10 应急措施

描述处理执行过程中发生的异常和其他时间所需要的各种活动。

测试意外事件报告模板

目录

1. 意外事件总结报告标识符
2. 意外事件总结

3. 意外事件描述

　3.1 输入

　3.2 期望得到的结果

　3.3 实际结果

　3.4 异常情况

　3.5 日期和时间

　3.6 规程步骤

　3.7 测试环境

　3.8 重现尝试

　3.9 测试人员

　3.10 见证人

测试日志模板

目录

1. 测试日志的标识符
2. 描述
3. 活动和事件条目

测试总结报告模板

目录

1. 测试总结报告标识符
2. 总结
3. 差异
4. 综合评估
5. 结果总结

　5.1 已解决的意外时间

　5.2 未解决的意外事件

6. 评价
7. 建议
8. 活动总结
9. 审批

参 考 文 献

[1] 朱少民. 软件测试方法和技术[M]. 北京：清华大学出版社，2005.

[2] 许健才. 从纵横两个方向谈软件测试的生命周期[J]. 大众科技，2011(2).

[3] 赵轶. 软件开发和软件生命周期[EB/OL]. http://www. 51testing. com，2007.

[4] 赵瑞莲. 软件测试[M]. 北京：高等教育出版社，2004.

[5] 杜文洁，景秀丽. 软件测试基础教程[M]. 北京：中国水利水电出版社，2008.

[6] 周元哲. 软件测试教程[M]. 北京：机械工业出版社，2010.

[7] Andreas Spillner，Tilo Linz，Hans Schaefer. 软件测试基础教程(第 2 版)[M]. 刘琴，周震漪，译. 北京：人民邮电出版社，2009.

[8] 佟伟光. 软件测试[M]. 北京：人民邮电出版社，2008.

[9] 聂林波，刘孟仁. 软件缺陷分类的研究[J]. 计算机应用研究，2004.

[10] Putnam Lawrence H，Myers Ware. Measures for Excellence：Reliable Software on Time，within Budget [M]. Prentice Hall，1992.

[11] 黄锡滋. 软件可靠性、安全性与质量保证[M]. 北京：电子工业出版社，2002.

[12] Ram Chillarege，Inderpal S Bhandari，Jarir K Chaar，et al. Orthogonal Defect Classification：A Concept for In-process Measurements [J]. IEEE Transactions on Software Engineering，1992，18(11).

[13] IEEE Std 1044-1993. IEEE Standard Classification for Anomalies[S].

[14] 朱少民. 如何正确理解自动化测试[EB/OL]. http://blog. csdn. net/kerryzhu/article/details/2958155.

[15] Rick D. Craig，Stefan P. Jaskiel. 系统的软件测试[M]. 杨海燕，等译. 北京：电子工业出版社，2003.

[16] 贺平. 软件测试教程[M]. 北京：电子工业出版社，2008.

[17] 刘力. 国内自动化测试现状及展望[EB/OL]. http://www. cesoo. info/？ p=179，2009.

[18] 张振兴. 浅谈软件测试自动化解决方案[J]. 软件测试，2007(2).

[19] IBM. Rational Robot[EB/OL]. http://www. ibm. com.

[20] HP. Winrunner，LoadRunner，QuickTest Professional，TestDirector [EB/OL]. http://www. hp. com.

[21] Compuware. QACenter[EB/OL]. http://www. compuware. com.

[22] Parasoft. Jtest[EB/OL]. http://www. parasoft. com.

[23] Mercury QuickTest Professional Tutorial，Mercury Interactive Corporation，2007.

[24] 领测软件. 软件测试工具中 QTP 功能测试流程[EB/OL]. http://www. ltesting. net/ceshi/ceshijishu/rjcsgj/mercury/quicktestpro/2010/0506/171228. html，2010.

[25] 陈能技. QTP 自动化测试实践[M]. 北京：电子工业出版社，2008.

[26] 飞思科技产品研发中心. 实用软件测试方法与应用[M]. 北京：电子工业出版社，2003.

[27] 王学集，林耀纳. phpwind[EB/OL]. http://www. phpwind. com. 2005.

[28] 冯莉. 面向对象的单元测试和集成测试技术研究[J]. 农业网络信息，2007(4).

[29] 王东刚. 软件测试与 JUnit 实践[M]. 北京：人民邮电出版社，2004.

[30] JUnit Tutorial. Lars Vogel Version 1. 4. 2011.

[31] Andrew Hunt,David Thomas. 单元测试之道 Java 版——使用 JUnit[M]. 陈伟柱,陶文 译. 北京：电子工业出版社,2005(1).

[32] 于涌. 精通软件性能测试与 LoadRunner 实战[M]. 北京：人民邮电出版社,2010.

[33] Mercury LoadRunner Tutorial, Version 8.1. 2005.

[34] 柳胜. 性能测试从零开始——LoadRunner 入门与提升[M]. 北京：电子工业出版社. 2011.

[35] 陈绍英,刘建华,金成姬. LoadRunner 性能测试实战[M]. 北京：电子工业出版社. 2007.

[36] Atlassian. JIRA[EB/OL]. http://www.atlassian.com/software/jira.

[37] Feigenbaum. A V. Total quality control [M]. New York: McGraw-Hill Book Company,1983.

[38] Powell T C. Total quality management as competitive advantage: A review and empirical study [J]. Strategic Management Journal, 1995, 16(1):15-37.

[39] Crosby P. Quality is free[M], McGraw-Hill, 1979.

[40] Roger S. Pressman. 软件工程：实践者的研究方法(第 4 版)[M]. 机械工业出版社：北京,1999.

[41] 胡铮. 软件测试与质量保证技术[M]. 科学出版社：北京, 2011.

[42] IBM. Implementing Software Inspections[M]. IBM course notes, 1981.

[43] Sauer C, Jeffery D R, Land, L., et al. The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research[J]. IEEE Transactions on Software Engineering. SE-26(1): 1-14.

[44] Tom Gilb D. Graham. Software Inspection[M]. Addison-Wesley Professional,1994.

[45] Edward Yourdon. StrucTrued Walkthroughs[M]. Yourdon,1978.

[46] 关河. 如何和开发工程师交流——给测试工程师的一点建议[EB/OL]. http://blog.csdn.net/dennis_duan/article/details/228070, 2004.